ioProgrammo Anno XI - N°3 (112)

ANTEPRIMA: JBUILDER 2007
LA NUOVA VERSIONE DELL'IDE DI BORLAND INCONTRA ECLIPSE. TUTTI I PARTICOLARI...

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90
VERSIONE STANDARD RIVISTA+CD €6,90

PER ESPERTI E PRINCIPIANTI
Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • MARZO 2007 • ANNO XI • N.3 (112)

SQL SERVER 2005: oltre il DATABASE

Ecco le tecniche per creare archittetture veloci, affidabili al servizio di applicazioni che non possono mai interrompersi

Integrazione con .NET
Estendi MS SQL Server con le DLL sviluppate in Visual Studio

Lavorare con XML
Esegui query ed immagazzina i dati usando direttamente il formato nativo

Gestione delle code
Diluisci il carico di lavoro distribuendo inserimenti e modifiche nel tempo

MEMORIA GESTITA CON C++

Entriamo nel cuore del compilatore e ti mostriamo come sfruttarne tutta la potenza per sviluppare software super ottimizzato...

MESSENGER QUIZBOT

Trasforma il tuo sistema di messaggistica in un gioco a premi. Crea una conversazione multipla con i tuoi amici. Vince chi fornisce più risposte C#

.NET FRAMEWORK
ARRIVA WPF/E: IL WEB DIVENTA ANIMATO
Grafica vettoriale, multimedia spinto, 3D. Usa subito tutto con le nuove tecnologie MS

JAVA
JAVA 6 INCONTRA JAVASCRIPT
Ecco come usare al meglio le funzionalità della nuova versione del compilatore di SUN

SISTEMA
VISUAL STUDIO LO FACCIO IO VB.NET
Rimpiazziamo i wizard standard e sviluppiamo un generatore di codice personalizzato

ARCHITTETTURE MODULARI
Creiamo un piccolo "Antivirus" utilizzando più linguaggi e mettendo tutto insieme

DATABASE
INFORMAZIONI IN MOVIMENTO
Ecco System Integration Services per importare dati in SQL Server da qualunque piattaforma

FATTI IL TUO ARCHIVIO C#
Usa il Windows Form Binding, la tecnica per "agganciare" i record all'interfaccia utente

REPORT LATO SERVER C#
Evitiamo che il sistema rimanga bloccato in attesa di una lunga elaborazione

- SQL SERVER BEST PRACTICE PARTE I E II
- SQL SERVER E XML

DELPHI
CREIAMO UN SERVIZIO WINDOWS
Usiamo Delphi, Firebird, e poco altro per scrivere un programma di gestione delle newsletter

PATTERN
LA MASCHERA DEL PROXY PYTHON
Ecco cosa fare se devi aggiungere una nuova funzione e non vuoi riscrivere tutto...

SOLUZIONI Quando non è possibile risolvere un problema con un un sì o con un no, arriva la "logica sfumata": fuzzy per gli amici

SQL SERVER 2005: OLTRE IL DATABASE • MEMORIA GESTITA CON C++ • MESSENGER QUIZBOT • JAVA 6 INCONTRA JAVASCRIPT
EDIZIONI MASTER

Anno XI - N.ro 03 (112) - Marzo 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, L. Buono, A. Galeazzi, F. Grimaldi, F. Smelzo, A. Pelleriti, M. Locuratolo, L. Corias
Segreteria di Redazione: Veronica Longo

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI

ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 21% sul prezzo di copertina di €75,90 · ioProgrammo con Libro (11 numeri) €75,90 sconto 30% sul prezzo di copertina di €108,90 Offerte valide fino al 31/03/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Servizio Abbonati:
☎ tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Febbraio 2007

Questo mese su ioProgrammo

▼ OLTRE IL DATABASE

C'è stato un tempo in cui la preoccupazione maggiore per un programmatore era quella di scrivere del buon codice. Routine performanti e ottime query SQL erano sufficienti per ottenere software di un certo spessore. Attualmente le cose sono radicalmente cambiate. Non è più sufficiente essere un buon programmatore per riuscire ad interagire con strutture complesse. Se avrete la pazienza di leggere l'articolo su SQL Server 2005 che presentiamo in questo numero vi accorgerete chea fianco ad ottime conoscenze di SQL e di un buon linguaggio di programmazione è necessario avere nozioni di sistemistica. E' il caso per esempio delle gestione delle code. Allo stesso modo se avrete la pazienza di leggere l'articolo sullo sviluppo di architetture modulari scoprirete come la conoscenza di più di un linguaggio di programmazione consente di sviluppare software maggiormente ottimizzato ed in minor tempo. Cosa è cambiato dunque? sicuramente sono cambiate le richieste delle aziende, ma anche e soprattutto nel tentativo di innalzare il livello tecnologico dei nostri software al contempo abbiamo aumentato il tempo della curva di apprendimento. Siamo dunque al paradosso di dover avere nozioni di sistemistica, conoscenza di più di una tecnica, e molto altro ma al contempo non disporre del tempo per imparare tutto questo. In realtà è un segno dei nostri tempi, dove lavorare in un mercato all'avanguardia come il nostro significa lavorare a ritmi decisamente elevati. Perciò assume un ruolo essenziale l'aggiornamento continuo e meticoloso, perché abbandonare per qualche tempo lo studio significa poi che difficilmente riusciremo a riempire il GAP che si verrà a creare fra noi e l'avanzare della tecnologia. Chi riuscirà ad aggiornarsi costantemente e con continuità avrà la capacità di tenere il passo con il mercato della programmazione. In tutti gli altri il rischio di rimanere emarginati diventa piuttosto elevato

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

SQL SERVER 2005: oltre il DATABASE

Ecco le tecniche per creare architeture veloci, affidabili al servizio di applicazioni che non possono mai interrompersi

Integrazione con .NET
Estendi MS SQL Server con le DLL sviluppate con Visual Studio

Lavorare con XML
Esegui Query ed immagazzina i dati usando direttamente il formato nativo

Gestione delle code
Diluisci il carico di lavoro distribuendo inserimenti e modifiche nel tempo

pag. 16

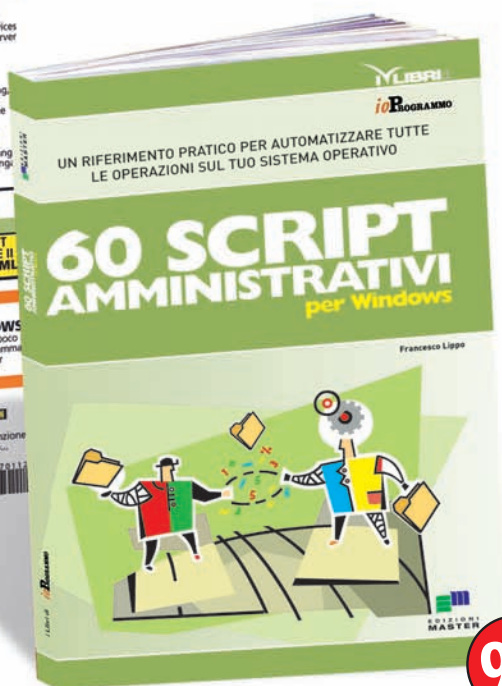
MEMORIA GESTITA CON C++

Entriamo nel cuore del compilatore e ti mostriamo come sfruttarne tutta la potenza per sviluppare software super ottimizzato

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

http://forum.ioprogrammo.it

RIVISTA + LIBRO + CD-ROM in edicola

I contenuti del libro

60 script amministrativi per windows

Quante volte vi è capitato di voler automatizzare questa o quell'operazione noiosa e ripetitiva? Si certo Windows è comodo con le sue interfacce grafiche. Ma il più delle volte è necessario interagire con il sistema per ottenere un risultato. Questo significa premere bottoni, dare e ottenere feedback continuamente e sottintende una presenza fisica continua accanto al computer. Ci sono invece operazioni che possono essere automatizzate semplicemente lanciando una stringa di comando e lasciando poi il computer ad elaborare per conto suo. Ce ne sono altre che possono partire in background e non richiedono la nostra presenza. Questo libro contiene circa sessanta esempi pratici che vi consentono di gestire al meglio il vostro sistema e rappresentano anche un esempio importante su cui basare eventuali successive personalizzazioni

UN RIFERIMENTO PRATICO PER AUTOMATIZZARE TUTTE LE OPERAZIONI SUL TUO SISTEMA OPERATIVO

- **Automazione del sistema e del file system**
- **Gestione dei servizi e delle applicazioni**
- **Condivisioni, connessioni e amministrazione della rete**

RIVISTA + CD-ROM in edicola

STRUTS 2.0

Il framework java per il pattern MVC

Il Model View Controller, è uno dei pattern maggiormente utilizzati in programmazione. Il suo scopo è semplice, ovvero dividere la struttura di un programma in tre moduli separati. Uno che si occupa della definizione delle classi di business ovvero il model, uno che si occupa di visualizzare i dati in output, ovvero la view, uno che si occupa di gestire il flusso dell'esecuzione. Un software che segue i dettami dell'MVC può dirsi ben strutturato, solido e facilmente manutenibile. Ora, mentre il pattern è unico, possono essistere diverse soluzioni applicative per la sua implementazione e non tutte efficaci. Struts mette a disposizione un framework ben strutturato ed ormai consolidato per la creazione di programmi Java aderenti al pattern MVC. Nel tempo è diventato un vero e proprio standard e pur conservando una certa complessità nessun programmatore Java che voglia sviluppare software professionale può fare a meno di aderirvi completamente. La versione 2 si presenta come profondamente rinnovata, molto più veloce della precedente, il codice è stato quasi completamente riscritto.

Utilizare questo framework significa mettersi nelle condizioni di scrivere software facilmente manutenibile, solido e aderente ad un pattern ormai consolidato

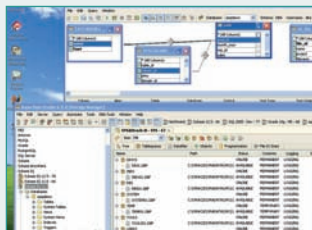
Directory: /struts

Prodotti del mese

Acqua Data Studio

Il SQL Manager è servito

Acqua Data Studio è uno strumento decisamente evoluto. Scritto in Java si configura come un SQL Manager completo e affidabile. Supporta un gran numero di DBServer, da Oracle a MsSQL a MySQL. Consente la gestione totalmente grafica delle tabelle, dei database, delle interrogazioni SQL. Si può usare in alternativa a PHPMyAdmin anche se è ancora leggermente inferiore per quanto riguarda il numero di funzionalitàesposte. D'altra parte PHPMy-Admin è un fuoriclasse nel suo genere ed è tarato su MySQL, mentre Acqua-Data Studio è del tutto generico. In ogni caso si tratta di un sistema che vi può trarre fuori dai guai in più di una situazione. Specialmente se volete un ambiente leggere da utilizzare. Da provare!

[pag.108]

Open Computer Vision Library 1.0

Ottimo per la grafica

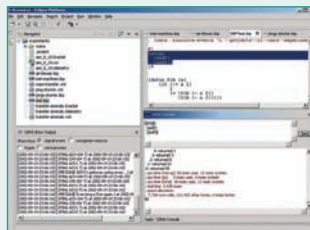
Più che una semplice libreria per la gestione degli oggetti grafici. Si tratta di un'insieme di API piuttosto futuristiche, si va da riconoscimento del movimento a quello facciale, all'isolamento dei contorni. Si tratta di una libreria immensa che offre un quantitativo sterminato di funzioni relative al multimedia e alla grafica. Su ioProgrammo ce ne siamo occupati spesso con progetti relativi al motion detection. In uno dei nostri articoli lo abbiamo utilizzato per creare un gioco di tiro al bersaglio elettronico, in cui la nostra libreria si occupava di intercettare un raggio di luce tramite una webcam posizionata dietro un disegno a cerchi concentrici e si preoccupava di assegnare un punteggio

[pag.109]

Eclipse SDK 3.2.1

L'IDE tuttofare

Eclipse è un progetto completo portato avanti da Eclipse Foundation con la collaborazione di una miriade di aziende fra cui IBM, Adobe, Sun e che si è prefissata lo scopo di creare un IDE estendibile per plugin adattabile a qualunque tipo di linguaggio o tecnologia. Di default Eclipse si propone come IDE per Java ed è qui che da il meglio di se. Ma proprio grazie ai suoi plugin è possibile utilizzarlo come ambiente di programmazione per PHP, per C++, per Flex e per molti altri linguaggi ancora. Inoltre sempre grazie per ciascun linguaggio sono disponibili altri plugin ad esempio per rendere l'ambiente RAD o per favorire lo sviluppo dei Web Services o altro. Insomma lo scopo è stato raggiunto completamente. Eclipse è realmente un IDE tuttofare, ormai maturo, e che serve una miriade di programmatori grazie alle sue caratteristiche di affidabilità e flessibilità. Unica nota negativa: una certa pesantezza che richiede un hardware di tutto rispetto

[pag.109]

EasyPHP 2.0b1

Web Server, Database e PHP tutto in un click

Sono in molti coloro che vorrebbero imparare PHP e che prima ancora che nel linguaggio trovano difficoltà nell'installazione di un ambiente di test per la programmazione. Per poter testare un software scritto in PHP è necessario avere installato almeno un Web Server ed il linguaggio stesso, opzionalmente è necessario MySQL. L'installazione di questi prodotti è per molti il primo ostacolo all'apprendimento di PHP. EasyPHP è un software All in Once che con pochi click vi consentirà di installare tutto il necessario senza avere alcuna conoscenza di sistemistica. Avrete un ambiente completo e funzionale integrato con mySQL, php e apache. A questo punto tutto quello che dovrete fare sarà concentrarvi sul codice

[pag.109]

GLI ALLEGATI DI IOPROGRAMMO

Questo mese tre Webcast dedicati a SQL Server 2005. Due "Best Practice" e una guida al nuovo formato "XML". Per sfruttarne subito e a fondo tutte le potenzialità.

SQL XML

SQL Server 2005 introduce il nuovo tipo di dato XML. Durante il webcast vengono illustrate le tecniche per interrogarlo, manipolarlo e indicizzarlo. Vengono inoltre analizzati pro e contro del suo utilizzo e possibili scenari applicativi.
Speaker: Andrea Benedetti.

SQL Server Developer Best Practice – Parte 1 e 2

SQL Server visto dalla parte dello sviluppatore. In questi webcast verranno date le risposte a tutte quelle domande che chi si pone chi sviluppa con SQL Server prima o poi si pone. Verranno mostrate una serie di best practices che definiscono i punti fissi dello sviluppo di soluzioni basate su SQL Server e verranno mostrate le tecniche per evitare gli errori più comuni risolvendo le problematiche legate al lavoro pratico di tutti i giorni. Oltre a questo si definiranno le linee guida per avere soluzioni performanti e di facile manutenzione, anche relativamente alla sicurezza.
Speaker: Davide Mauri.

msdn WEBCAST

PERCORSI FORMATIVI

Per chi desidera approfondire sono disponibili sul sito di Microsoft una serie di strumenti dedicati a SQL Server e ai suoi componenti Reporting Services • SQL Server 2005: SQL Server Broker • SQL Server 2005: .NET Integration • SQL Server 2005: .NET Coding (UDF, UDT, Trigger, ecc.) • SQL Server 2005 • tecnologie per la disponibilità dei dati • SQL Server 2005: Tools and Add-ins

Visita

http://www.microsoft.com/italy/msdn/webcast

FAQ

Cosa sono i Webcast MSDN?

MSDN propone agli sviluppatori una serie di eventi gratuiti online e interattivi che approfondiscono le principali tematiche relative allo sviluppo di applicazioni su tecnologia Microsoft. Questa serie di "corsi" sono noti con il nome di Webcast MSDN

Come è composto tipicamente un Webcast?

Normalmente vengono illustrate una serie di Slide commentate da un relatore. A supporto di queste presentazioni vengono inserite delle Demo in presa diretta che mostrano dal vivo come usare gli strumenti oggetto del Webcast

Come mai trovo riferimenti a chat o a strumenti che non ho disponibili nei Webcast allegati alla rivista?

La natura dei Webcast è quella di essere seguiti OnLine in tempo reale. Durante queste presentazioni in diretta vengono utilizzati strumenti molto simili a quelli della formazione a distanza. In questa ottica è possibile porre domande in presa diretta al relatore oppure partecipare a sondaggi etc. I Webcast riprodotti nel CD di ioProgrammo, pur non perdendo nessun contenuto informativo, per la natura asincrona del supporto non possono godere dell'interazione diretta con il relatore.

Come mai trovo i Webcast su ioProgrammo

Come sempre ioProgrammo cerca di fornire un servizio ai programmatori italiani. Abbiamo pensato che poter usufruire dei Webcast MSDN direttamente da CD rappresentasse un ottimo modo di formarsi comodamente a casa e nei tempi desiderati. Lo scopo tanto di ioProgrammo, quanto di Microsoft è infatti quello di supportare la comunità dei programmatori italiani con tutti gli strumenti possibili.

Su ioProgrammo troverò tutti Webcast di Microsoft?

Ne troverai sicuramente una buona parte, tuttavia per loro natura i webcast di Microsoft vengono diffusi anche OnLine e possono essere seguiti previa iscrizione. L'indirizzo per saperne di più è: http://www.microsoft.it/msdn/webcast/msdn segnalo nei tuoi bookmark. Non puoi mancare.

L'iniziativa sarà ripetuta sui prossimi numeri?

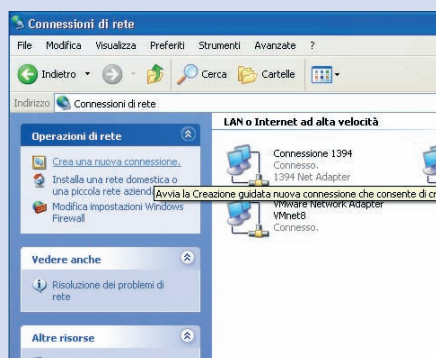
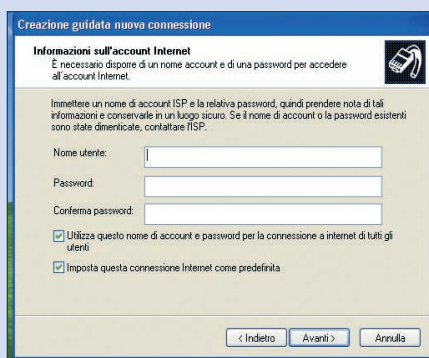
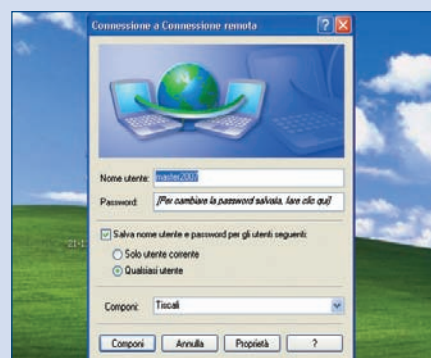
Sicuramente si.

Online con Tiscali Easy

Numero unico per tutta l'Italia e nessuna registrazione. Il modo più semplice e veloce per entrare in Internet risparmiando

Sei spesso lontano da casa e hai necessità di collegarti a Internet ovunque ti trovi? Desideri salvaguardare la tua privacy navigando in modo anonimo? Non hai voglia di perdere tempo in lunghe registrazioni per creare un nuovo account? Niente paura, Tiscali ha pensato anche a te! Con Tiscali Easy arriva un sistema tutto nuovo di collegarsi a Internet. Non sarà più necessario creare un nuovo account e fornire i propri dati personali, potrai navigare collegandoti con un unico numero di telefono (7023456789) da tutta Italia e soprattutto utilizzando i dati di accesso forniti direttamente da Tiscali (UserID e Password presenti sulla scheda), quindi non collegabili in alcun modo a te. Insomma, con Tiscali Easy, otterrai in un colpo solo riservatezza dei dati, risparmio del tempo necessario alla creazione di un abbonamento e tariffe vantaggiose. I costi di connessione sono simili a quelli di un classico abbonamento gratuito: 1,90 cent. per i primi 10 minuti e 1,72 cent. per quelli successivi. Per risparmiare ulteriormente è possibile connettersi a Internet durante le ore serali o nei giorni festivi al costo di 1,09 cent. per i primi dieci minuti e 0,98 cent. per quelli successivi. L'unico requisito richiesto per poter eseguire la connessione è un modem correttamente installato. Poiché si tratta di una normale connessione analogica è possibile utilizzare i tool di accesso remoto integrati in Windows

IN RETE CON TISCALI

Nessuna registrazione basta creare una nuova connessione e inserire i dati di accesso forniti da Tiscali

1 NUOVA CONNESSIONE Portiamoci nel pannello di controllo, e selezioniamo l'icona connessioni di rete. In alto a sinistra selezioniamo "nuova connessione" e prepariamoci a seguire il wizard che comparirà

2 DATI DI ACCESSO Seguiamo il Wizard fino a quando non ci vengono chiesti i dati per l'autenticazione. E' sufficiente inserire quelli presenti nella card allegata a questo numero di ioProgrammo: master2007/tiscali

3 ACCESSO A INTERNET Connettersi è semplicissimo, troveremo una nuova icona all'interno del pannello di controllo alla voce connessioni. Bastano due click ed il gioco è fatto sarete connessi ovunque vi troviate

News

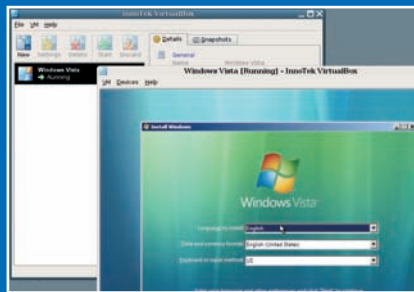
MICROSOFT RILASCIA ASP.NET AJAX

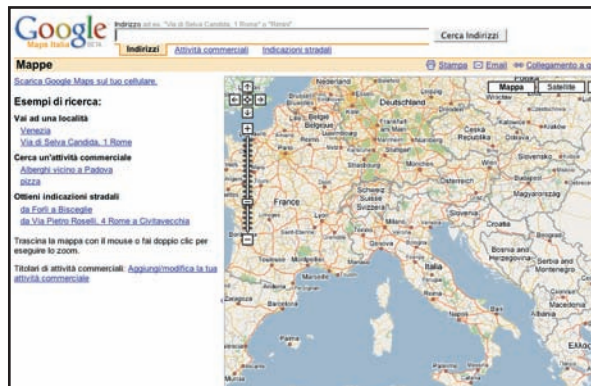
Anticipando decisamente i tempi Microsoft ha appena rilasciato il suo framework per integrare la tecnologia ASP.NET in Ajax. In realtà il prodotto già da tempo si trovava in uno stato di beta testing piuttosto avanzato e già da tempo le varie versioni se pur non ufficiali avevano fatto capolino fra i vari siti sviluppati in ASP.NET. Il rilascio della versione ufficiale segna da un lato il consolidamento di questo strumento verso ambienti di produzione e dall'altro sottolinea la volontà di Microsoft di investire in una tecnologia che in pochissimo tempo sta radicalmente cambiando il modo di pensare al Web

IBM VA VERSO JAVA 6

Big blue ha appena rilasciato un proprio SDK per Java 6. Il prodotto può essere scaricato all'indirizzo https://www14.software.ibm.com/iwm/web/cc/earlyprograms/ibm/java6/?ca=dgr-lnxw01IBM-SDK-Java-v6. Le piattaforme supportate sono praticamente tutte. Le novità risiedono soprattutto nella direzione della stabilità e delle performance. IBM ha focalizzato la propria attenzione verso alcuni strumenti di diagnostica e backtracing. Si tratta di un passo importante per IBM che affianca così ad una vasta gamma di strumenti server anche un prodotto per sviluppatori che può godere del supporto diretto di un'azienda che copre l'infrastruttura IT di moltissimi enti ed istituzioni. Daltra parte non è la prima volta che qualcuno attenta alla leadership di Sun

VIRTUALBOX SFIDA VMWARE

Così dopo XEN, VmWare subisce l'attacco di un altro agguerrito concorrente. Questa volta si tratta di VirtualBox, prodotto innovativo rilasciato recentemente da Innotek. Si tratta ovviamente di un software di virtualizzazione. I sistemi supportati sono praticamente tutti incluso il recentissimo Windows Vista.
Funziona sia su macchine Linux su sistemi Microsoft e rispetto ai concorrenti si differenzia per la leggerezza, la facilità di installazione e sopratutto per la licenza. Il prodotto è disponibile in due versioni, con licenza GPL per chi scarica direttamente i sorgenti dal CVS, e con licenza PUEL (Personal Use ed Evalutation License) per chi scarica direttamente i binari. Nella seconda modalità è ovviamente possibile utilizzarlo pienamente solo per scopi personali e di valutazione, nonostante questo il prodotto è pienamente fruibile. Le uniche differenze di rilievo rispetto alla versione OpenSource risiedono nell'impossibilità di condividere cartelle fra sistema host e remoto e nel supporto alle periferiche USB. Si tratta di un software molto ben concepito che sicuramente farà ancora parlare di se nel mercato della virtualizzazione che come in molti avranno notato, sempre più sta diventando terreno di scontro per tutti e al quale puntano colossi del calibro di Microsoft, VmWare ma anche piccoli ma agguerriti come Xen e adesso VirtualBox

PHP ASSALTA LE GOOGLE MAPS

Google ci ha abituato ormai ad una serie di servizi innovativi che non mancano di appassionare noi programmatori. Fra i tanti c'è quello delle Google Maps, ovvero il servizio che consente di visualizzare una mappa geografica di un luogo a partire dalle sue coordinate oppure da un indirizzo. Questo servizio si basa su due pilastri fondamentali: il Web e Javascript. Se per il primo non c'è niente da osservare è invece piuttosto scomodo dovere ricorrere a Javascript da linguaggi come PHP ad esempio. A darci una mano è intervenuta recentemente System Seven Design che ha appena rilasciato Phoogle ovvero una classe PHP appositamente studiata per interfacciarsi al meglio con le API di google maps. L'uso della libreria è davvero facile e intuitivo, si tratta di appena una dozzina di funzioni, la curva di apprendimento è davvero bassa

PREOCCUPAZIONE PER XAML

E' appena arrivato sul mercato e già desta una serie di preoccupazioni e polemiche, si tratta di XAML. Il nuovo formato che costituisce una parte importante di Windows Vista per ciò che concerne la rappresentazione delle informazioni, ha suscitato la reazione dell'ECIS (European Committee for Interoperable Systems). Secondo l'ECIS, XAML sarebbe progettato in modo da dare l'assalto all'HTML e isolare in questo modo gli utenti non Windows dall'utilizzo della Rete. Scenario da fantascienza aggiungiamo noi, ma facilmente leggibile attraverso le parole di Simon Awde, presidente dell'ECIS il quale sostiene che la massiccia presenza di XAML in Windows Vista porterebbe lontano gli autori dei siti Web dall'utilizzo degli standard aperti su cui fino ad ora il Web si è basato. E tuttavia Ecis non sarebbe da solo a ritenersi preoccupato ma agirebbe sotto precise segnalazioni fornite da colossi del calibro IBM, Sun, Nokia, Adobe, Oracle e Red Hat. L'introduzione di una tecnologia quale XAML porta sempre e comunque un minimo di scompiglio all'interno di un mercato difficile come quello della rete. Tuttavia oltre alla definizione formale di uno standard a fare il successo di una tecnologia è sempre stata la sua capacità di essere utile per un utente. E' sempre stato così, fin dagli albori, ed a maggior ragione sul web. Linguaggi come PHP, python, Ruby e lo stesso HTML spopolano sul Web da anni senza godere del budget di marketing dei concorrenti

IL PDF VA VERSO LA CERTIFICAZIONE

Il noto formato di Adobe è ormai da anni uno standard de facto per lo scambio di qualunque tipo di documento. Tuttavia la differenza fra uno standard de facto e uno standard riconosciuto e certificato dall'ISO è davvero notevole. Il peso di questa differenza comincia a farsi sentire quando enti e istituzioni chiedono con forza di potere accedere alle specifiche. Da qui la decisione di Adobe di sottoporre il formato PDF alla standardizzazione ISO. Come sempre accade, al di la delle dichiarazioni di affetto nei confronti dei formati aperti, è seguita qualche polemica. In molti hanno fatto notare infatti come l'annuncio di Adobe sia arrivato dopo il lancio da parte di Microsoft di un formato che dovrebbe rivaleggiare con PDF, ovvero l'XPS. In realtà Adobe già molto prima aveva iniziato le procedure di standardizzazione per alcuni dei formati che fanno capo al PDF

ORACLE VICINO A MYSQL

BigO da tempo a iniziato ad investire risorse ed energie nei mercati opensource. Così dopo avere annunciato il rilascio della propria distribuzione Unbreakable Linux basata su Red Hat enterprise ed avendola arricchita con una serie di tool per la gestione centralizzata, è arrivato adesso il momento di offrire il proprio supporto pe r il database più usato su sistemi opensource: MySQL. Per ora si tratta solo di indiscrezioni ma Marten Mickos CEO di MySQL avrebbe lasciato trapelare notizia di un contatto di MySQL AB con Oracle che porterebbe proprio BigO a supportare direttamente MySQL. Non si tratta di una notizia da poco se si pensa che insieme Oracle e Mysql detengono una fetta consistente del mercato dei database se pur su segmenti decisamente diversi

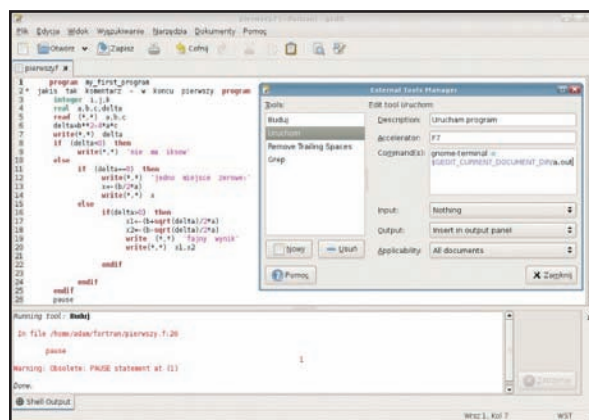
LE BOTNET AFFOSSANO INTERNET

A sostenerlo è Vint Cerf il padre del protocollo TCP/IP che oggi occupa un posto di rilievo come consulente di Google. La sua preoccupazione questa volta non è relativa al quantitativo di indirizzi internet ormai assegnati e che sono vicini al limite massimo teorico di quelli consentiti dal TCP/IP. Questa volta i suoi pensieri sono rivolti alle Botnet, ovvero computer infetti da virus e malware di ogni genere e che in un qualche modo si aiutano a vicenda nel propagarsi su altre macchine. Lo scenario descritto da Cerf diventa ancora più apocalittico quando parla di SpamThru, ovvero un Super Trojan in grado di infettare milioni di PC ma anche di controllare gli eventuali malware già presenti su questi ultimi. L'esito di questa operazione sarebbe la creazione di un supervirus in grado di sferrare un attacco "letale" a qualunque azienda o organizzazione presente su Internet. Se a tutto questo si aggiungono le preoccupazioni reali costituite dalla presenza della CyberMafia su Internet che appunto sfrutta le debolezze dei sistemi per tenere sotto scacco organizzazioni di qualunque tipo, si capisce bene di quale importanza siano gli avvertimenti di Vint Cerf. Evitare completamente questi pericoli è però virtualmente impossibile. Ci si attendono spiragli di luce da Windows Vista ma soprattutto l'accortezza degli utenti può più di ogni altra cosa evitare che la rete diventi terreno di caccia per organizzazioni del tutto illegali

PRONTO RUBY ON RAILS 1.2

Il pluripremiato framework per lo sviluppo Internet basato sul linguaggio Ruby giunge così alla versione 1.2. Non si tratta di uno stacco importante con la precedente versione, tuttavia il codice generato adesso risulta molto più pulito, in particolar il numero di controlli è stato diminuito dai precedenti 15 ad appena 2 o 3 facilitando così la curva di apprendimento. Altre modifiche sono state introdotte alle API. Il rilascio della prima versione di Ruby On Rails ha segnato un deciso salto in termini di approccio alla programmazione, tanto che aziende dal nome ben più blasonato si sono affannate ad imitarne lo stile. Questo rilascio è nel segno della continuità, il prodotto si appresta a diventare maturo e a ricoprire un'area importante della programmazione web

SUN PROGETTA IL SUO FORTRAN

Il linguaggio più amato dai matematici potrebbe non essere per niente morto. Sun crede ancora nelle sue potenzialità ed in tal senso ha aperto un laboratorio di ricerca dedicato al Fortress le cui specifiche sono disponibili all'indirizzo http://research.sun.com/projects/plrg/. Si tratterebbe di un linguaggio molto simile al Fortran connotato per una forte propensione alla matematica. Una prima versione binaria dell'interprete e del compilatore è già disponibile e si registra un discreto fermento delle community rispetto a questo progetto, segno evidente di quanto un buon prodotto se pur a distanza di anni rimane comunque tecnologicamente utile. D'altra parte il Fortran nonostante la sua età non teme rivali nella sua capacità di elaborazione matematica

SQL SERVER 2005 OLTRE IL DATABASE

NON SOLO SQL PER IL DB DI MICROSOFT, MA SERVIZI CHE CONSENTONO DI REALIZZARE ARCHITETTURE COMPLESSE. GETTEREMO LE BASI PER CREARE UN SOFTWARE DI GESTIONE DELLE RICARICHE TELEFONICHE...

SQL Server 2005 è un prodotto ricchissimo di funzionalità. Ogni funzionalità, presa singolarmente, rappresenta una novità e un grosso punto di forza dell'intera piattaforma. Basti pensare, ad esempio, ai nuovi livelli di isolamento "snapshot", al database mirroring, ai trigger DDL e via dicendo. Dal punto di vista strettamente legato allo sviluppo un'importanza fondamentale rivestono le novità riguardanti il supporto nativo ad XML, l'integrazione con il .NET Framework e un notevole e potente sistema di gestione asincrona delle code.
Tutte queste feature possono essere usate da sole, ma la potenza e la flessibilità che offrono sono ancora più apprezzabili quando vengono fatte lavorare in modo coordinato ed integrato; è in questo modo che si può sfruttare al 100% la potenza dell'intera piattaforma. E' questo il modo in cui, nel presente articolo, andremo ad approfondire e mettere alla prova tre di queste importanti feature: supporto nativo ad XML, integrazione con .NET ed il Service Broker.

LO SCENARIO

Lo scenario in cui immaginiamo di dover operare è uno scenario distribuito, dove una serie numerosa di client deve comunicare in tempo reale informazioni al server centrale, in modo da poter realizzare quello che si può definire un "on-line business". Rendiamo ancora più pratico l'esempio. Supponiamo che il cliente che stiamo seguendo sia una grossa azienda di telefonia mobile. Si deve implementare un'architettura per la gestione delle ricariche telefoniche. Tali ricariche devono poter essere effettuate tramite i metodi classici: card prepagata da "grattare" ed invio del codice tramite SMS, sito web della propria banca, sito web ad-hoc della compagnia telefonica o sportello bancomat.
L'architettura prevede che ognuno dei metodi suddetti si appoggi ad uno client studiato appositamente per la piattaforma di riferimento. Tale client comunica al server centrale, i dati dell'utente e della transazione. I dati, prima di poter essere utilizzati – e quindi procedere con l'accredito della somma richiesta - devono essere validati e memorizzati su un database SQL Server. Il primo problema da affrontare è piuttosto chiaro: il server centrale avrà un carico di lavoro estremamente disomogeneo, anche durante una singola giornata. In alcuni momenti ci saranno da evadere poche richieste di ricarica, ed altri in cui le richieste saranno cosi tante da poter potenzialmente affondare il server. Quest'ultima situazione è evidentemente da evitare a tutti i costi, in quanto il disservizio creato sarebbe enorme, con pesanti ripercussioni sul profitto del nostro cliente. Il secondo problema è invece legato alla possibilità di acquistare del credito telefonico tramite l'invio di un codice via SMS. Tale codice deve essere evidentemente "complesso" per evitare che un utente possa indovinarne uno, ed inoltre non deve avere una logica o un algoritmo di generazione intuibile e replicabile dall'utente finale in quanto quest'ultimo potrebbe altrimenti generarsi i codici da solo. Ora che abbiamo un quadro completo della situazione (che è ovviamente solamente una semplificazione della realtà) possiamo cercare di capire come risolvere tutto questo con SQL Server e le nuove funzionalità che mette a disposizione. Per rendere l'esempio praticabile nello spazio di un articolo ci limiteremo a gestire l'accredito via SMS. Gli obiettivi che ci prefiggiamo sono quindi:

- Gestione degli SMS in arrivo (per semplicità si suppone che l'SMS in arrivo sia già processato e trasformato in un XML).
- Validazione del codice contenuto nell'SMS.
- Memorizzazione della transazione.

Il tutto, ovviamente, tenendo conto che non possiamo mai trovarci nella situazione in cui il server è così oberato di richieste da generare errori di time-out.

INTEGRAZIONE CON .NET

SQL Server 2005 permette di scrivere oggetti il cui funzionamento è definito non utilizzando T-SQL ma un qualsiasi linguaggio .NET.
Stored procedure, trigger, funzioni possono ora essere scritte sfruttando al 100% il .Net Framework e le sue enormi potenzialità. Oltre a questi oggetti "classici" è possibile anche scrivere nuove funzioni di aggregazione (Custom Aggregates) e nuovi tipi di dati. Grazie a queste feature si riescono ad ottimizzare notevolmente le prestazioni e si aprono nuovi scenari architetturali. Nel caso in cui fossimo nella necessità di dover operare sui dati in modo "complesso" (ossia più complesso di quello che il semplice codice T-SQL ci permetterebbe di fare) finalmente possiamo fare tutto all'interno di SQL Server sfruttando comunque le potenzialità di un linguaggio potente ed evoluto. Il tutto senza dover esser costretti ad effettuare il solito oneroso e dispendioso procedimento di estrazione-elaborazione-importazione dei dati verso un'applicazione esterna.
Nel progetto legato a questo articolo la necessità di usare il .NET Framework la troviamo nella richiesta di verificare la validità del codice di accredito in modo che sia univoco e con caratteri di controllo (un po' come il codice fiscale), in modo da esser sicuri che non sia possibile da parte di entità esterna allo società di telefonia, "indovinare" o generare codici di accredito validi, e quindi poter bypassare l'acquisto del credito stesso.
Tale algoritmo è piuttosto semplice ma complesso da realizzare con T-SQL: dato un numero casuale di 22 cifre, si sommano tutti le cifre che occupano una posizione pari nel numero tra loro, e del numero ottenuto si prendono le ultime due cifre dello stesso. Si ottiene così una sequenza di 24 cifre.
La cosa molto interessante dell'utilizzo di oggetti managed (ossia scritti in .Net) è che dall'esterno sono visti come oggetti standard di SQL Server e quindi la loro invocazione ed il loro utilizzo non è in alcun modo diverso dai reciproci oggetti scritti in T-SQL. Questo ci permetterà di poter utilizzare la semplicità del codice T-SQL e, laddove questo non sia sufficientemente potente e performante per le procedure di controllo che dobbiamo implementare, utilizzare .NET senza però dover mai "uscire" da SQL Server. Creare un oggetto tramite .NET è piuttosto semplice. Una volta scelto cosa creare i books online di SQL Server spiegano per filo e per segno quali sono le regole secondo la quale dobbiamo implementare la classe .Net che incapsula il funzionamento dello stesso. Una volta compilato ed ottenuto l'assembly è sufficiente caricare quest'ultimo in SQL Server utilizzando il comando CREATE ASSEMBLY e poi associarlo all'oggetto T-SQL vero e proprio (che quindi fungerà da wrapper) utilizzando l'opzione EXTERNAL NAME del comando DDL "CREATE".
Un esempio completo di come poter importare in SQL Server due funzioni managed che saranno utilizzate nel progetto è il seguente

```
use SMSDemo
go

-- Importa l'assembly contenente le funzioni
create assembly CreditCodeValidator
from 'D:\Work\Solid
 Quality\IoProgrammo\ArticoloSqlServer2005\CreditCo
 deValidator\CreditCodeValidator\bin\Debug\CreditCod
                                      eValidator.dll'
go

-- Verifica l'importazione
select * from sys.assemblies
go

-- Crea la funzione ValidateCreditCode
CREATE FUNCTION
          [dbo].[ValidateCreditCode](@creditCode
                                     [nvarchar](24))
RETURNS [bit] WITH EXECUTE AS CALLER
AS
EXTERNAL NAME
        [CreditCodeValidator].[UserDefinedFunctions].
                                [ValidateCreditCode]
go

-- Crea la funzione GenerateCreditCode
CREATE FUNCTION [dbo].[GenerateCreditCode]()
RETURNS [nvarchar](24) WITH EXECUTE AS CALLER
AS
EXTERNAL NAME
        [CreditCodeValidator].[UserDefinedFunctions].
                                [GenerateCreditCode]
GO

-- Testa le funzioni:

-- Genera un codice
select dbo.GenerateCreditCode();

-- Verifica la correttezza di un codice
-- (0=non valido, 1=valido)
-- Il codice 415747001177177725465568
-- deve risultare valido
select
dbo.ValidateCreditCode(
                    '415747001177177725465568');
```

SUPPORTO XML NATIVO

Il supporto ad XML, in realtà, è presente in SQL Server sin dalla versione 2000. Con la versione 2000, però, il supporto è limitato al comando OPENXML

ed all'opzione FOR XML.
OPENXML permette di accedere ad un documento XML (o a porzioni di esso) come se fosse una tabella, cosi da poter estrarre e manipolare i valori in esso contenuti usando i canonici comandi T-SQL, oppure per poter mettere relazione lo stesso, tramite join ad esempio, con oggetti relazionali (tabelle, viste o query).
L'opzione FOR XML, invece, permette di generare XML. Utilizzata come opzione del comando SELECT, infatti, permette di generare non un resultset classico (righe e colonne), ma un documento XML.
Il problema più grosso che rimane aperto con SQL Server 2000, però, è il fatto che non è possibile memorizzare XML in quanto tale nelle colonne di una tabella, ne è possibile utilizzarlo senza effettuare il cast verso il tipo di dato (n)varchar. Questo significa, però, perdere tutte le funzionalità intrinseche di XML, e quindi l'impossibilità di validare lo stesso tramite un XML Schema; inoltre si perde anche la possibilità di effettuare query XPath (o XQuery) e quindi diventa molto difficile prelevare o ricercare determinati valori all'interno del documento stesso.
SQL Server 2005 pone rimedio a tutte queste limitazioni. In SQL Server 2005, infatti, è stato introdotto il tipo di dato XML che può essere utilizzato sia per definire una colonna, il tipo di dato di una variabile o di un parametro.
Oltre a questo sono stati introdotti dei comandi specifici per la gestione e la manipoliazione dei documenti XML cosi memorizzati. Non solo è possibile effettuare ricerche tramite XPath/XQuery ma è anche possibile modificare l'XML stesso direttamente tramite i comandi T-SQL, estesi per l'occasione.

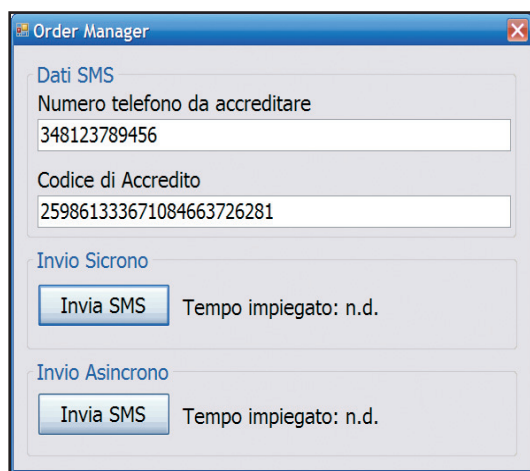



Fig. 1: Uno screenshot dell'applicazione test

Un esempio pratico di come poter utilizzare XML è riportato nel seguente listato

```
use SMSDemo;
go

-- Dichiaro una variabile di tipo XML
declare @x xml;

-- Assegno un documento XML alla variable.
set @x = '<sms>
          <creditRequest>
               <phone number="348123789456" />
              <code value=
                    "996320478780987912003437"
               />
          </creditRequest>
          </sms>';
-- Tramite il metodo "value"
-- è possibile estrarre il
-- valore dal documento XML

select phone_number =
 @x.value('(/sms/creditRequest/phone/@number)[1]',
                                        'bigint'),
  code =
     @x.value('(/sms/creditRequest/code/@value)[1]',
                                     'varchar(24)')
```

Una domanda potrebbe a questo punto sorgere spontanea. Come si colloca XML all'interno di un database relazionale? Un database poggia le sue basi sul concetto di tuple ed attributi (righe e colonne) ed per il suo funzionamento ottimale le regole di normalizzazione sconsigliano fortemente l'utilizzo di valori non atomici (prima forma normale), ossia dati composti da insiemi di valori più semplici. Esempi di dati non atomici sono stringhe di valori separati da virgola oppure proprio i documenti XML; per loro natura, infatti, questi ultimi contengono diverse informazioni organizzate in modo gerarchico.
E' bene quindi sottolineare, quindi, che l'utilizzo di XML in un database come SQL Server è ideale, e davvero molto interessante, come mezzo di trasporto dei dati. Potendo utilizzare il tipo di dato XML come parametro di funzioni o stored procedure diventa molto comodo passare ad una stored procedure, ad esempio, un documento XML contenente tutti i dati di cui abbiamo bisogno (che magari sono molti e non sempre quantificabili a priori. Si pensi ad esempio ad un ordine: se è vero che la testata dello stesso è ben definita e conosciuta, non è altrettanto vero per il numero di righe che, essendo variabili, non possono essere facilmente passate come parametri) e quindi gestirli in un sol colpo e comodamente utilizzando i nuovi comandi dedicati ad XML. È in questo modo – come mezzo di trasporto – che utilizzeremo XML nel nostro progetto, passandolo cioè come paremetro ad una srored procedure per la successiva elaborazione.

SERVICE BROKER

Il Service Broker è un servizio interno di SQL Server 2005 (non va quindi installato a parte) che permette l'invio asincrono di messaggi ad altri Service Broker anche remoti.
Questo elemento è fondamentale per poter realizzare il progetto che ci siamo preposti. Un vincolo importante è infatti quello che il server, anche sotto pesanti carichi di lavoro, non si debba mai trovare nella condizione di posporre cosi tanto l'esecuzione di una query da far andare in time-out quest'ultima o, peggio, di bloccarla per un periodo di tempo indefinito, che poi magari può sfociare più facilmente in un deadlock e quindi nell'annullamento della transazione.
Serve pertanto un meccanismo diverso dai classici comandi di SELECT, INSERT, UPDATE o DELETE che non sia come quest'ultimi sincrono ma sia invece asincrono. In pratica sarebbe ideale poter inviare un messaggio al server dove richiediamo che vengano fatte una serie di operazioni del tipo descritto in precedenza, ad esempio una INSERT, ma senza che questo debba essere fatto immediatamente se il server è già sovraccarico ed inoltre senza che per questo tale attesa debba essere per noi bloccante.
Ricordo, infatti, che tutti i comandi di SQL sono sincroni, ossia una volta in esecuzione la connessione che li ha inviati non può fare altro che aspettare il termine dell'esecuzione del comando stesso e solo ad operazione avvenuta può inoltrare un'altra richiesta al server.
Il Service Broker, invece, permette l'invio di richieste al server che verranno evase immediatamente se il server è sufficientemente scarico e può quindi iniziare l'esecuzione del comando, oppure le mette in coda al fine di non sovraccaricare eccessivamente l'RDBMS, evandendole non appena in grado di farlo.
La cosa particolarmente interessante è che il Service Broker gestisce in modo completamente automatico la coda di richieste ed è in grado anche di scalare in automatico in modo da poter svuotare la stessa il più velocemente possibile.
Una caratteristica importante di questo sistema è che le richieste che possono essere inviate al Service Broker non sono comandi SQL, ma sono messaggi XML che possono essere presi in carico da una stored procedure attivata automaticamente dal broker.
Il tutto, essendo all'interno di un RDBMS (non dimentichiamolo!) in modo completamente transazionale.
Questo ci permette quindi di realizzare un'architettura asincrona, non bloccante, in grado di raccogliere tutte le richieste di accredito anche nelle ore di punta, senza sovraccaricare il server, senza richiedere l'implementazione manuale di un sistema di code o lo sfruttamento di un sistema di esterno che comunque dovrebbe essere integrato con la nostra soluzione, con un dispendio di energie e di risorse notevolmente più alto.
Il Service Broker, come avrete già intuito, è uno strumento estremamente potente e flessibile. Come ogni strumento con queste caratteristiche, ha anche una configurazione piuttosto complessa. Per rendere le cose più semplici in questo articolo ci limiteremo ad implementare una soluzione basata sul Service Broker in modalità "locale". In pratica il broker invierà e gestirà messaggi che entrano ed escono dallo stesso database, cosi da sfruttare la sua capacità di gestione asincrona delle code, che è poi la cosa che ci interessa. In realtà il Service Broker può tranquillamente funzionare anche tra server remoti, scambiandosi messaggi attraverso la rete, criptando ed autenticando i messaggi stessi. La cosa, però, non è banale e pertanto, nel caso vogliate provare, vi invito come prima cosa a prendere visione dei tutorial che ho scritto sull'argomento [3].

	id	receiveddate	orderdata	process_result
1	1	2007-01-28 21:34:40.840	<sms><creditRequest><phone number="348123789456...	Ok
2	2	2007-01-28 21:34:47.233	<sms><creditRequest><phone number="348123789456...	Codice già usato
3	3	2007-01-28 21:34:52.233	<sms><creditRequest><phone number="348123789456...	Codice già usato
4	4	2007-01-28 21:35:21.217	<sms><creditRequest><phone number="348123789456...	Codice non valido
5	5	2007-01-28 21:35:44.483	<sms><creditRequest><phone number="348123789456...	Codice non valido

Fig. 2: ***Un esempio di select su campi XML***

Gli oggetti vitali per il funzionamento del broker sono:

- Message Types
- Contracts
- Queues
- Services

I Message Types definiscono che tipo di messaggi possono essere scambiati dal broker. Nel caso il messaggio sia di tipo XML si può definire uno schema che il documento XML deve rispettare.
I Contracts stabiliscono quali messaggi possono essere inviati da chi invia e quali da chi riceve.
Le Queues sono il contenitore fisico dei messaggi in arrivo. Sono accedibili tramite una semplice SELECT ed il loro contenuto viene mostrato sottoforma di tabella.
I Services sono ciò che permettono alle queues di ricevere i messaggi. In pratica fungono da livello di astrazione tra la coda ed il mondo esterno.
Definiti questi importanti oggetti, è possibile avere esempio pratico di configurazione del service broker nel listato seguent

```
use SMSDemo;
go

-- Creiamo il message type che useremo
-- come mezzo di trasporto dei dati.
-- Il messaggio dovrà essere well-formed xml
```

NOTE

APPROFONDIRE

[1] Quando utilizzare XML in un database?
(http://blogs.ugidotnet.org/nettools/articles/14330.aspx)

[2] Service Broker Tutorial – Parte 1
(http://www.ugiss.org/articoli.aspx?mid=376&ctl=ArticleView&articleId=97)

[3] Service Broker Tutorial – Parte 2
(http://www.ugiss.org/articoli.aspx?mid=376&ctl=ArticleView&articleId=144)

[4] User Group Italiano di SQL Server
(http://www.ugiss.org)

[5] SQLCLR: estendere le funzionalità con .Net
(http://www.microsoft.com/italy/msdn/risorsemsdn/sql/sqlclr.mspx)

```
create message type [SMSCreditRequestSubmission]
                              validation = well_formed_xml;

-- Creiamo il contratto (molto semplice)
-- che definisce  che la richiesta
-- di accredito sarà inviata dal client server
-- e non viceversa

create contract [SMSCreditContract]
(
   [SMSCreditRequestSubmission] sent by initiator
);

-- Creiamo i due oggetti Queue che
-- fungeranno da "caselle postali"
create queue [SMSCreditRequestSubmissionQueue];
create queue [SMSCreditManagentQueue];
```

```
-- Creiamo i due servizi che permetteranno
-- alle code di ricevere i
-- messaggi che invieremo

create service [SMSCreditRequestService] on queue
                    [SMSCreditRequestSubmissionQueue]
(
   [SMSCreditContract]
);
create service [SMSCreditManagementService] on
                      queue [SMSCreditManagentQueue]
(
   [SMSCreditContract]
);
```

Configurato il Service Broker, è possibile inviare e ricevere messaggi come mostrato nel listato 4 (mittente) e nel listato 5 (destinatario).

```
use SMSDemo;
go

-- inizio transazione
begin transaction

-- inizio dialogo
declare @conversationHandle uniqueidentifier;

begin dialog conversation
            @conversationHandle
from service
            [SMSCreditRequestService]
to service
            'SMSCreditManagementService'
on contract
            [SMSCreditContract]
with
            encryption = off,
            lifetime = 600;
-- invio messaggio
send on conversation
            @conversationHandle
message type
            [SMSCreditRequestSubmission] (N'<order>'
        + cast(getdate() as nvarchar(50)) + '</order>')

-- fine conversazione (utilizzo della tecnica Fire &
            Forget SOLO PER SEMPLIFICARE L'ESEMPIO)
end conversation @conversationHandle

commit
go

-- I dati vanno da [SMSCreditRequestService] a
                        [SMSCreditManagementService],
-- e quindi dalla coda
        [SMSCreditRequestSubmissionQueue] alla coda
                            [SMSCreditManagentQueue]

-- Se tutto è andato bene la coda del mittente inviati
                                           sarà vuota
select * from
           dbo.[SMSCreditRequestSubmissionQueue];

-- La coda di destinazione, invece, conterrà due
                                            messaggi.
-- Il primo è il messaggio vero e proprio, il secondo è
                                        un messaggo di
-- sistema che indica che il dialogo è stato terminato
                                          dal mittente.
select * from dbo.[SMSCreditManagentQueue];
select cast(message_body as xml) from
                      dbo.[SMSCreditManagentQueue];
```

E' importante sottolineare che attualmente il Service Broker supporta dialoghi e non monologhi. Questo significa che ad ogni messaggio inviato deve corrispondere un messaggio di risposta…un po' come il "passo e chiudo" che ci si comunica vicendevolmente quando si comunica utilizzando walkie-talkie. Non è quindi possibile (in realtà tecnicamente è possibile ma è fortemente sconsigliato) implementare tecniche di comunicazione unidirezionale tipo "fire-and-forget" (ossia monologhi).

IL PROGETTO

Definiti tutti gli elementi necessari per implementare il nostro progetto, è ora di definirne l'architettura di alto livello:

- un'applicazione winform simula l'invio di un SMS contenente un codice di accredito, producendo un documento XML che viene inviato al Service Broker
- Il Service Broker si prende in carico del messag-

gio, dandolo in input ad una stored procedure preposta che:

- Preleva i dati dal messaggio XML;
- Verifica la validità dei dati e del codice di accredito tramite l'utilizzo di una funzione scritta con .NET;
- Scrive i dati della transazione nel database, incrementando il credito dell'utente;

I file completi per la creazione del progetto sono sono prelevabili dal sito http://www.davidemauri.it/ioprogrammo-esempiosql.zip.

Rispetto a quanto descritto fino ad ora, il progetto contiene non una, ma tre stored procedure. Com'è possibile vedere nel *listato 6*,

```
use SMSDemo;
go

create procedure
        [dbo].[stpSlowSMSHandlingProcedure]
@orderData as xml
as
declare @result varchar(1000)

set xact_abort on
begin tran
-- Simuliamo una transazione molto lunga

waitfor delay '00:00:15';

declare @phone_number bigint
declare @code char(24)

select
        @phone_number =
@orderData.value('(/sms/creditRequest/phone/@num
                ber)[1]', 'bigint'),
        @code =
@orderData.value('(/sms/creditRequest/code/@value)
                [1]', 'char(24)')

if (dbo.ValidateCreditCode(@code) <> 0)
begin
        -- Memorizza il valore del credito
        declare @amount money
        select
                @amount = credit
        from
                dbo.CreditCodes
        where
                credit_code = @code
        and
                available <> 0

        if (@@rowcount > 0) begin
                -- Aggiorna il credito dell'utente
                update
                        dbo.Users
                set
                        credit = credit +
                                @amount
                where
                        phone_number =
                                @phone_number

                -- Imposta il codice come
                                utilizzato
                update
                        dbo.CreditCodes
                set
                        available = 0
                where
                        credit_code = @code
                set @result = 'Ok'
        end else begin
                set @result = 'Codice già usato'
        end
end else begin
        set @result = 'Codice non valido'
end

insert into dbo.SMS (orderdata, process_result)
        values (@orderData, @result);
commit
```

il codice della stored procedure "stpSlowSMS HandlingProcedure" usata dal progetto contiene un comando WAITFOR, che la rende lenta in modo artificiale. Questa forzatura ci permette di apprezzare come la scalabilità del sistema sia gestita in modo autonomo dal Service Broker.
La stored procedure "stpSlowSMSHandling ProcedureWithBroker" è invece la stored procedure utilizzata per ottenere un funzionamento asincrono. Nel listato 7 si nota come non ci sia nessun riferimento alla stored procedure menzionata in precedenza, che processa effettivamente la richiesta di accredito. In questa procedura, infatti, viene solamente inviato un messaggio XML ad una coda del Service Broker.

```
use SMSDemo;
go

create procedure
    [dbo].[stpSlowSMSHandlingProcedureWithBroker]
@orderData as xml
as

-- begin tran
begin transaction
```

```
-- begin dialog
declare @conversationHandle uniqueidentifier;

begin dialog conversation
            @conversationHandle
from service
            [SMSCreditRequestService]
to service
            'SMSCreditManagementService'
on contract
            [SMSCreditContract]
with
            encryption = off,
            lifetime = 600;

-- send message
send on conversation
            @conversationHandle
message type
            [SMSCreditRequestSubmission]
                                        (@orderData)

-- end conversation
-- ATTENZIONE! Per semplicità si sta facendo uso di
                          una tecnica tipo fire-and-forget
-- ma la chiusura della conversazione dovrebbe
                          avvenire solo dopo che il ricevente
-- ha ricevuto il messaggio ed ha richiesto di
                          terminare la conversazione, in quanto
-- non ha bisogno di ricevere altri messaggi per poter
fare il suo lavoro.
end conversation @conversationHandle

commit
```

È il Service Broker stesso, alla ricezione del messaggio, ad invocare la procedura di gestione della richiesta di accredito. Questa possibilità è data dalla funzionalità conosciuta come "Activation", che permette di associare ad una coda una stored procedure che verrà automaticamente invocata all'arrivo di un messaggio nella coda stessa:

```
alter queue dbo.[SMSCreditManagentQueue] with
activation
            (status = on,
procedure_name =
 dbo.stpHandleIncomingSMSCreditRequestSubmission,
max_queue_readers = 15,
execute as self);
```

Il valore max_queue_readers informa il broker del massimo numero di esecuzioni contemporanee possibili della stored procedure specificata in procedure_name. Il Service Broker, infatti, se verifica che i messaggi in entrata nella coda sono più di quelli che riescono ad essere evasi (perché la stored procedure ha un lungo tempo di esecuzione), provvende in automatico ad attivare altre stored procedure che provvederanno ad eleborare i messaggi della coda, in modo da poterla svuotare nel più breve tempo possibile, senza per questo affondare il server, o far attendere eccessivamente l'elaborazione dei dati.
Cliccando più volte sul pulsante "Invio SMS" nella sezione "Invio Sincrono" sull'interfaccia dell'applicazione di esempio (figura 1), si nota come utilizzando la stored procedure in modo classico (e quindi sincrono) l'applicazione rimanga bloccata fino al termine dell'esecuzione della stessa, impedendo l'invio di altri SMS.
Cliccando invece sull'omonimo pulsante nella sezione "Invio Asincrono", che invece fa uso del Service Broker, l'interfaccia non rimane mai bloccata: le richieste di esecuzione vengono memorizzate nella coda messaggi ed elaborate in modo automatico ed asincrono.
Cliccando più volte sul pulsante sarà sempre possibile continuare ad inviare SMS. Facendo questa operazione molte volte nell'arco di un minuto si può simulare un carico di lavoro particolarmente alto.
Diventa così possibile apprezzare, attraverso la vista di sistema sys.dm_broker_activated_tasks, la gestione automatica della scalabilità del broker; effettuando infatti la query

```
select * from sys.dm_broker_activated_tasks
```

si possono vedere quante sono le stored procedure attivate in automatico dal Service Broker per poter evadere il più velocemente possibile le richieste in coda. Quest'ultima è visibile usando questa query:

```
select * from [SMSSubmissionQueue]
```

L'applicazione di esempio, una volta processato il messaggio, aggiornerà la tabella SMS e la tabella USERS. Nella prima si potranno vedere i risultati dall'elaborazione dei messaggi arrivati (figura 2), mentre nella seconda si potrà vedere il credito dell'utente aumentare in funzione del codice utilizzato (i codici sono presenti nella tabella CREDITCODES).

CONCLUSIONI

Il progetto mette in luce le grandi potenzialità dell'intera piattaforma SQL Server 2005. Solo due anni fa una soluzione come quella proposta avrebbe necessariamente richiesto molti più sforzi per ottenere un risultato sotto molti punti di vista inferiore. Per ovvi motivi l'esempio è stato abbondantemente semplificato- E' possibile approfondire sia tramite i Books Online di SQL Server sia anche tramite le risorse web che trovate nella sezione "Approfondimenti" dell'articolo.

Davide Mauri

L'AUTORE

Davide Mauri è un consulente specializzato nell'implementazione, nello sviluppo e nell'ottimizzazione di soluzioni basate su .NET , SQL Server 2005 e delle tecnologie ad essi collegate (Integration Services, Analysis Services, Notification Services, Reporting Services). Oltre alla consulenza si dedica alla formazione ed alla divulgazione sempre in ambito .NET e SQL Server 2005, erogando corsi e partecipando a conferenze nazionali ed internazionali. Nel 2006 è stato insignito del premio MVP Award per SQL Server da Microsoft per le sue conoscenze e il suo supporto alla community di SQL Server. Dal 2007 è presidente di UGISS, il primo e più importante User Group Italiano di SQL Server. Dal 2007 è socio e mentor della divisione italiana di Solid Quality Learning Per contattarlo:
dmauri@solidquality learning.com
http://blogs.ugidotnet.org/nettools

IL MIGRATORE DI DATI UNIVERSALE

PROBLEMA: UN VOSTRO CLIENTE VI CHIEDE DI SVILUPPARE UN NUOVO PRODOTTO PER LA GESTIONE DEL CARICO/SCARICO DEL MAGAZZINO SENZA PERDERE I DATI CHE HA GIÀ. COME FATE AD INTEGRARE IL VECCHIO FORMATO CON IL VOSTRO NUOVO LUCCICANTE SOFTWARE?

Chi lavora con i database si trova spesso a dover affrontare problemi relativi all'importazione di dati da sorgenti esterne al database, o di manipolazione degli stessi per aggregare dei dati o alterarli. Con SQL Server 7, Microsoft ha introdotto uno strumento per l'ELT chiamato DTS. DTS permettono di estrarre dati da diverse fonti, manipolarli e trasferirli in delle destinazioni. Le fonti e le destinazioni possono essere di diversa natura: dei database, dei file ASCII o dei fogli di calcolo; anche le trasformazioni possono essere di varia natura, come ad esempio delle aggregazioni o dei calcoli sui valori. I DTS forniscono un set di oggetti programmabili tramite cui è possibile effettuare le operazioni di ETL sopra descritte e tanto altro ancora.

SQL SERVER INTEGRATION SERVICES

SQL Server Integration Services (che nel corso dell'articolo chiameremo SSIS) è la nuova piattaforma di Business Intelligence di Microsoft. Sostituisce i DTS, estendendone notevolmente le funzionalità. Gli utilizzi di SSIS vanno dal Data Warehousing all'ETL, alla manutenzione dei database. La piattaforma SSIS è stata completamente ridisegnata sia dal punto di vista dell'interfaccia grafica verso l'utente sia da quello architetturale e funzionale. Il risultato è una piattaforma programmabile ed estendibile, con un ambiente di sviluppo completo ed estremamente potente grazie ad un insieme molto ricco di componenti.
La novità forse più interessante l'introduzione di due motori separati per il flusso di controllo e per il flusso dei dati. Il flusso di controllo viene gestito dal "motore di run-time", il quale si occupa di attività come la gestione delle transazioni, delle variabili, del debug etc. Il flusso dei dati è gestito dal "motore del flusso dei dati" che si occupa dei processi di trasformazione dei dati.

INSTALLAZIONE

SSIS è fornito con SQL 2005 a partire dalla versione standard. Per installarlo è sufficiente selezionare il pacchetto "Integration Services" dalla maschera dei componenti durante il setup.
Nel corso dell'installazione viene creato un servizio sotto l'account "Network" ed un software per la gestione dei package chiamato "Business Intelligence Developer Studio".

BUSINESS INTELLIGENCE DEVELOPMENT STUDIO

Business Intelligence Development Studio (che in seguito chiameremo BIDS) è l'ambiente di sviluppo per i package SSIS; una versione di Visual Studio customizzata per lavorare con SQL Server 2005. L'ambiente di sviluppo eredita tutte le funzionalità di Visual studio; è infatti possibile organizzare i package in progetti e soluzioni, effettuarne il debug o utilizzare degli strumenti per il controllo delle versioni. Vediamo come si presenta l'ambiente di sviluppo. In basso abbiamo le Connections; dove possiamo configurare le connes-

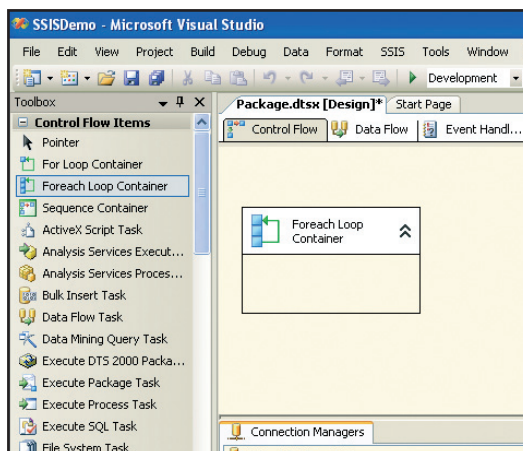



Fig. 1: Business Intelligence Development Studio

sioni alle sorgenti di dati ed alle destinazioni. Sono presenti un discreto numero di tipi di connessione, da sorgenti ADO, Excel, Flat File ed atri ancora. Sulla sinistra abbiamo la toolbox con gli oggetti disponibili per la creazione del package:

- **Data Sources:** sono le possibili sorgenti di dati
- **Transformations**: le trasformazioni
- **Data Destinations:** le destinazioni

Sulla abbiamo destra l'explorer della soluzione. Al centro dello schermo possiamo notare che il package è suddiviso in 4 sezioni:

- Control Flow
- Data Flow
- Event Handlers
- Package Exporer

I primi tre ne rappresentano il nucleo, mentre l'explorer serve a navigare comodamente tra oggetti e proprietà. In base alla sezione in cui stiamo lavorando, la toolbox viene popolata con degli elementi diversi. La sezione "***Control Flow***" consente di gestire il motore di run-time. In questa sezione troviamo i componenti per la gestione del flusso di esecuzione. Vi troviamo componenti per l'esecuzione di script SQL, ActiveX o sul FileSystem. Tra le novità più interessanti troviamo i cicli for e foreach, che consentono di eseguire attività ripetitive su degli oggetti. La sezione "***Data Flow***" consente di gestire il motore di flusso dei dati. Gli elementi sono suddivisi in tre categorie: sorgenti, trasformazioni e destinazioni ed in tutte e tre le categorie troviamo nuovi ed interessanti componenti. Gli "Evevt Handlers" consentono infine di definire dei gestori per eventi come ad esempio l'errore sull'esecuzione del package. È possibile associare i gestori di evento a qualsiasi livello: dal singolo task, fino al package. Gli eventi vengono gestiti attraverso dei package secondari, dando all'utente uno strumento molto potente.

UN ESEMPIO CONCRETO

Per il nostro esempio utilizzeremo un database di una società che vende dei prodotti on-line.
In questo database sono presenti dei prodotti suddivisi in categorie e sottocategorie; altre informazioni sul prodotto sono prezzo e dimensioni.
Abbiamo poi i clienti catalogati per regione e provincia.
Quando un cliente effettua un ordine, questo viene salvato nella tabella **OrdersHeader**, mentre la lista dei prodotti acquistati è salvata nella tabella **OrdersDetail**.

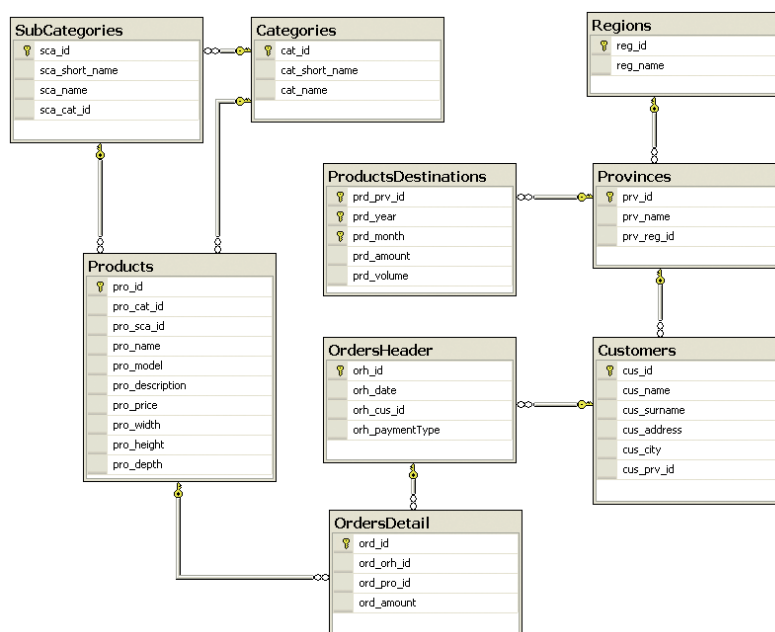



Fig. 2:* *Il database dei prodotti

Ecco gli script per la loro creazione:

```
USE [Products]
GO

CREATE TABLE [dbo].[OrdersHeader](
        [orh_id] [int] NOT NULL,
        [orh_date] [datetime] NOT NULL,
        [orh_cus_id] [int] NOT NULL,
        [orh_paymentType] [char](3) NOT NULL
)
GO

CREATE TABLE [dbo].[OrdersDetail](
        [ord_id] [int] NOT NULL,
        [ord_orh_id] [int] NOT NULL,
        [ord_pro_id] [int] NOT NULL,
        [ord_amount] [int] NOT NULL
)
GO
```

Chi gestisce le spedizioni, per organizzare la logistica, potrebbe voler sapere quante spedizioni vengono fatte mensilmente in ogni provincia e qual è l'ingombro del materiale spedito.
Queste informazioni possono essere reperite mediante una query.

```
SELECT
 YEAR(orh_date) anno, MONTH(orh_date) mese,
 prv_name provincia,
 SUM(ord_amount) quantita,
SUM(ord_amount*(pro_width*pro_height*pro_depth
                                    )) volume
FROM ordersheader
```

```
INNER JOIN ordersdetail
    ON orh_id = ord_orh_id
INNER JOIN customers
    on orh_cus_id = cus_id
INNER JOIN provinces
    on cus_prv_id = prv_id
INNER JOIN products
    on ord_pro_id = pro_id
GROUP BY
  YEAR(orh_date),
  MONTH(orh_date),
  prv_id, prv_name
ORDER BY
 YEAR(orh_date),
  MONTH(orh_date)
```

Non è molto complicata, ma presenta un problema, per esempio nel caso di un sito molto visitato che produce 100.000 ordini in un giorno in tutta Italia con clienti che acquistano, in media, 3 prodotti. La tabella ordersdetail degli ordini di un mese conterrebbe 9.000.000 di tuple; volendo fare l'estrazione dei dati di un anno ci troveremmo a fare i conti con quasi 110.000.000 tuple.
Perché quindi non consolidare i dati in una nuova tabella in cui memorizziamo solo le informazioni di cui abbiamo bisogno? La tabella **ProductsDestinations** contiene le informazioni su: anno, mese, regione, quantità di prodotti consegnati e volume occupato da tali prodotti.
Il compito del nostro pacchetto sarà quello di leggere i dati sulle spedizioni, aggregarli e salvarli nella nostra tabella.
Il vantaggio ?
In Italia abbiamo 110 province, se moltiplichiamo per 12, in un anno avremo al massimo 1320 tuple su cui fare le nostre interrogazioni.

Creiamo la tabella delle destinazioni

```
USE [Products]
GO
CREATE TABLE [dbo].[ProductsDestinations](
            [prd_prv_id] [int] NOT NULL,
            [prd_year] [int] NOT NULL,
            [prd_month] [int] NOT NULL,
            [prd_amount] [int] NULL,
            [prd_volume] [float] NULL
)
```

Anche la query che legge i risultati sarebbe molto più semplice

```
SELECT
  prd_year,
  prd_month,
  prv_name,
  prd_amount,
  prd_volume
FROM ProductsDestinations
INNER JOIN Provinces ON prd_prv_id = prv_id
ORDER BY prd_year, prd_month
```

CREIAMO IL PACCHETTO

Se apriamo Visual Studio e clickiamo "File->New Project", possiamo notare che, dopo l'installazione di SQL Server 2005, è stata aggiunta una nuova categoria di progetti: "***Business Intelligence Projects***". Il template in questione servirà per dare una mano ai nostri scopi.
Creiamo un "Integration Services Project".

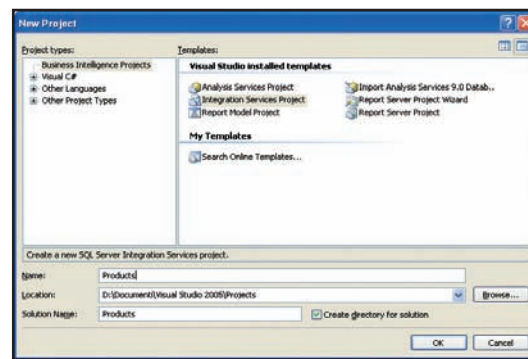



Fig. 3: Creiamo il progetto

Definiamo come prima cosa la nostra sorgente dati. Nell'area di lavoro principale, in basso troviamo una zona "Connection Manager" dedicata alle connessioni. Cliccando nell'area con il tasto destro del mouse, possiamo vedere una serie di tipi di connessione che BIDS ci mette a disposizione.
Selezioniamo "New OLE DB Connection" ed inseriamo le informazioni relative al database.
Iniziamo con il posizionare nell'area "**Control Flow**" un "*Execute SQL Task*"; lo utilizzeremo per scrivere un log di avvio del nostro package.

Clicchiamo con il tasto destro del mouse sull'oggetto e selezioniamo "Edit…" per impostarne le proprietà. Diamogli un nome, impostiamo la connessione al database dei prodotti e scriviamo come istruzione SQL:

```
INSERT INTO SSISLogs(ssi_log) values ('Start')
```

Una volta scritto il log di avvio, il package leggerà quattro parametri - anno e mese di inizio, anno e mese di fine - e farà un ciclo per ogni mese trovato per aggiornare le quantità relative a quel mese.
Creiamo quindi alcune variabili che ci serviranno per l'esecuzione del package. Nella parte sinistra dell'ambiente di lavoro, accanto alla toolbox, troviamo un tab dedicato alla definizione delle variabili.

Creiamo le seguenti variabili

- *yearFrom*: Int32; anno inizio importazione
- *yearTo*: Int32; anno fine importazione
- *monthFrom*: Int32; mese inizio importazione
- *monthTo*: Int32; mese fine importazione
- *monthsToLoop*: Object; la lista dei mesi da ciclare
- *year: Int32*; l'anno corrente all'interno del ciclo
- *month: Int32*; il mese corrente all'interno del ciclo

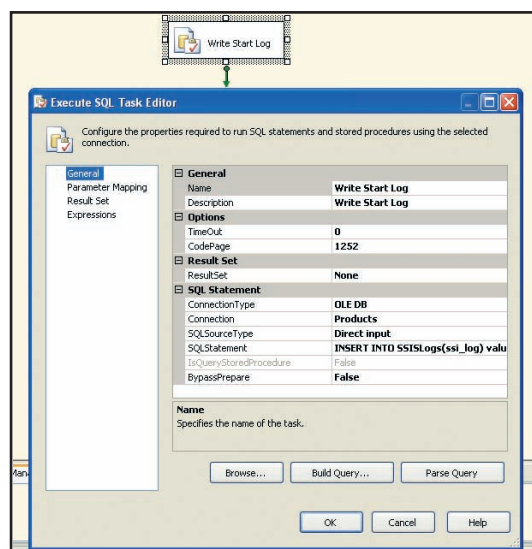



Fig. 5: Lo Start Log

Il primo step sarà leggere la lista dei mesi con degli ordini all'interno del range selezionato. Aggiungiamo un altro "*Execute SQL Task*", clicchiamo su "*Edit*" ed impostiamo nome e connessione. Passiamo quindi alla parte di recupero dei dati. Come query, utilizziamo la seguente:

```
SELECT DISTINCT
  YEAR(orh_date), MONTH(orh_date)
FROM OrdersHeader
WHERE
  YEAR(orh_date) BETWEEN ? AND ?
  AND MONTH(orh_date) BETWEEN ? AND ?
ORDER BY
  YEAR(orh_date), MONTH(orh_date)
```

Notiamo dei punti interrogativi al posto di mesi ed anni. Sono le variabili da mappare nella sezione "*Parameter Mapping*", come possiamo vedere in figura.
Resta un'ultima cosa da fare: configurare l'output. Per farlo, passiamo alla sezione "*Result Set*" ed aggiungiamo un nuovo set associandolo alla variabile monthsToLoop che abbiamo precedentemente definito.

Variables

Name	Scope	Data Type	Value
yearFrom	ProductsDestinations	Int32	2006
monthTo	ProductsDestinations	Int32	10
year	ProductsDestinations	Int32	0
yearTo	ProductsDestinations	Int32	2006
monthsToLoop	ProductsDestinations	Object	System.Object
monthFrom	ProductsDestinations	Int32	10
month	ProductsDestinations	Int32	0

Fig. 6: Le variabili del package

A questo punto occorre ciclare la collection ottenuta e, per ogni mese trovato, fare i calcoli sulle quantità e salvare i dati nella nostra tabella "*ProductsDestinations*".
Ci viene in aiuto il "**Foreach Loop Container**", un nuovo elemento di SSIS molto utile quando si devono eseguire operazioni ripetitive su delle collection.
Clickiamo con il tasto destro del mouse sul container e selezioniamo "Edit…"
Nel tab "*Collection*", impostiamo come Enumerator un "*Foreach ADO Enumerator*" e, come sorgente dati, la variabile "*montshToLoop*".
Nella sezione "*Variable Mappings*" selezioniamo le nostre "*year*" e "*month*".
Per completare la parte di "**Control Flow**", aggiungiamo un semplice step "*Execute SQL Task*" per scrivere il log di fine esecuzione come abbiamo fatto per quello di inizio esecuzione.

Passiamo quindi alla definizione del "Data Flow". Questa parte del package dovrà:

General
Parameter Mapping
Result Set
Expressions

Variable Name	Direction	Data Type	Parameter Name
User::yearFrom	Input	SHORT	@yearFrom
User::yearTo	Input	SHORT	@yearTo
User::monthFrom	Input	SHORT	@monthFrom
User::monthTo	Input	SHORT	@monthTo

Fig. 7: Mappiamo i parametri del task

- leggere i valori relativi al mese e all'anno del ciclo corrente
- effettuare le aggregazioni dei dati
- salvare i dati aggregati nella tabella di destinazione

Selezioniamo dai "*Control Flow Items*" un "*Data Flow Task*" e trasciniamolo dentro il ciclo Foreach. Facciamo doppio clic sul task per spostarci sul Data Flow corrispondente.
Come possiamo notare sulla sinistra, nella toolbox sono cambiati gli elementi; non abbiamo più quelli relativi al Control Flow, ma quelli propri del Data Flow.
Il primo elemento che andiamo ad inserire è una

POSSIBILI ERRORI

Eseguendo il comando *xp_cmdshell* potremmo ricevere un errore del genere:

Msg 15281, Level 16, State 1, Procedure xp_cmdshell, Line 1 SQL Server blocked access to procedure 'sys.xp_cmdshell' […]

Nel caso dovesse succedere, occorre impostare un parametro nella "*Surface Area Configuration for Features*" per consentire l'esecuzione della stored *xp_cmdshell*

sorgente dati. Selezioniamo dalla toolbox un "***OLE DB Source***" e spostiamola nell'area del Control Flow. Per configurarla, clicchiamo con il tasto destro del mouse sull'oggetto e selezioniamo "Edit…" Nel "*Connection Manager*" selezioniamo la sorgente dati, come "*Data Access Mode*" scegliamo "*SQL Command*" e come query inseriamo la seguente:

```
SELECT
  YEAR(orh_date) anno,
  MONTH(orh_date) mese,
  prv_id,
  ord_amount,
  (pro_width*pro_height*pro_depth)*ord_amount
                                        prd_volume
FROM ordersheader
INNER JOIN ordersdetail
    ON orh_id = ord_orh_id
INNER JOIN customers
    on orh_cus_id = cus_id
INNER JOIN provinces
    on cus_prv_id = prv_id
INNER JOIN products
    on ord_pro_id = pro_id
WHERE
   YEAR(orh_date) = ?
   AND MONTH(orh_date) = ?
ORDER BY prv_id
```

I punti interrogativi sono le variabili. Queste vanno definite in modo posizionale ed inserite cliccando su "Parameters" accanto alla textbox in cui inseriamo la query; inseriamo year come prima variabile e month come seconda.
Nella sezione "*Columns*", scegliamo tutte le colonne proposte.
Una volta letti i dati, utilizziamo il componente "*Aggregate*" dei Data Flow per aggregare i dati raggruppando gli ordini in base a provincia, mese ed anno.
Clicchiamo con il tasto destro del mouse sull'aggregate, selezioniamo "Edit…".
Selezioniamo le colonne disponibili ed impostiamo le aggregazioni come in figura:

-prv_id: Group By
-anno: Group By
-mese: Group By
-ord_amount: Sum
-prd_volume: Sum

A questo punto non ci resta che salvare i dati nella tabella di destinazione!
Prima di farlo, però, effettuiamo un semplice test. La tabella *ProductsDestinations* ha come chiave [prd_prv_id, prd_year, prd_month]; se cercassimo di inserire i valori per un mese gia processato, l'esecuzione della query andrebbe in errore. Per evitare questo inconveniente, utilizziamo un "***Lookup***" per testare se abbiamo già inserito la tupla nella tabella.
Nelle proprietà del Lookup scegliamo la connessione e la tabella *ProductsDestinations* per il test. Nel tab "*Columns*" mappiamo le colonne di input con quelle della tabella di destinazione.
Se il Lookup restituisce dei risultati, vuol dire che la riga è stata inserita, altrimenti no. Nel caso in cui la riga non sia già stata inserita, effettuiamo l'insert; in caso contrario un update. Effettuiamo l'update invece di uscire perché questo consente di riprocessare gli ordini di un mese per cui il calcolo è stato già fatto.
Aggiungiamo quindi un "**OLE DB Command**" per l'inserimento dei dati aggregati nella tabella di destinazione.
Andiamo in edit sul componente per configurarlo. Nel primo tab, il "*Connection Manager*", specifichiamo la connessione al database.
Nel secondo, impostiamo il nome e la query per l'insert.

```
INSERT INTO ProductsDestinations
(

    prd_prv_id,
    prd_year,
    prd_month,
    prd_amount,
    prd_volume
)
VALUES
(?, ?, ?, ?, ?)
```

Infine, tramite i tab "Column Mappings", mappiamo le colonne di input con i parametri della query (che al solito sono gestiti in modo posizionale).
Il componente per l'update viene configurato allo stesso modo, cambiano solo la query che è la seguente:

```
UPDATE ProductsDestinations
SET
  prd_amount = ?,
  prd_volume = ?
WHERE
   prd_prv_id = ?
  AND prd_year = ?
  AND prd_month = ?
```

ed il mapping delle colonne, in quanto adesso i primi due parametri sono *prd_amount* e *prd_volume*.
A questo punto, il nostro package è quasi pronto per l'esecuzione.
Prima di vederlo in azione, introduciamo breve-

mente la terza sezione dell'area di lavoro: "**Event Handlers**" in cui inseriamo un gestore per gli errori. Selezioniamo "*OnTaskFailed*" come Event Handler ed inseriamo un "*Execute SQL Task*" con la query *INSERT INTO SSISLogs(ssi_log) values ('Error')* per loggare il fatto che c'è stato un errore.

Deploy

Compiliamo il package e copiamo il file "*ProductsDestinations.dtsx*" presente nella directory bin del progetto in "*C:\SSISDeploy*"

Eseguiamo quindi il deploy del package sul nostro server tramite l'utilità dtutil. Apriamo una *commad prompt* ed eseguiamo:

```
dtutil /FILE " C:\SSISDeploy\ProductsDestinations.dt
                    sx" /DestServer Nome_Server /COPY
                            SQL;ProductsDestinations
```

ESECUZIONE

Una volta installato il package siamo pronti per eseguirlo. Possiamo farlo tramite uno script T-SQL

```
DECLARE @yearFrom varchar(4),
              @yearTo varchar(4),
              @monthFrom varchar(2),
              @monthTo varchar(2),
              @command varchar(1000)

SET           @yearFrom = '2006'
SET           @yearTo = '2006'
SET           @monthFrom = '10'
SET           @monthTo = '10'

SET @command  = 'dtexec /SQL
    "\ProductsDestinations" /SERVER "cri\sql2005" ' +
                                     '/MAXCON
           CURRENT " -1 " /CHECKPOINTING OFF ' +
                                     '/SET "\
      Package.Variables[User::yearFrom].Properties
                   [Value]";' + @yearFrom + ' ' +
                                     '/SET "\
  Package.Variables[User::yearTo].Properties[Value]";'
                              + @yearTo +  ' ' +
                                     '/SET "\
    Package.Variables[User::monthFrom].Properties
                  [Value]";' + @monthFrom + ' ' +
                                     '/SET "
     \Package.Variables[User::monthTo].Properties
                      [Value]";' + @monthTo + ''
EXEC xp_cmdshell @command
```

Abbiamo dichiarato ed impostato le variabili da settare sul package.

Tramite la ***xp_cmdshell***, eseguiamo il comando ***dtexec*** passandogli, tramite l'opzione /SET, come parametri, il nome del package che abbiamo installato e le variabili del package da impostare durante l'esecuzione.

CONCLUSIONI

SSIS è uno strumento molto più potente e flessibile del suo predecessore (i DTS).

Il suo ruolo non è soltanto quello di strumento delegato alla migrazione dei dati da un formato ad un altro ma può essere utilizzato in tutte quelle occasioni dove sia necessario effettuare una qualunque operazione che coinvolga l'elaborazione sequenziale di un formato. Si pensi per esempio a tecniche di normalizzazione.

SSIS può essere utilizzato con successo per integrre basi di dati preesistenti con un nuovo formato. La velocità di elaborazione e la presenza di un'interfaccia grafica sufficientemente intuitiva garantisce sicuramente un abbattimento dei tempi di produzione.

Certamente è necessario entrare in sintonia con il quantitativo notevole di funzionalità messe a disposizione da SSIS, ma u na volta superato il gradino iniziale sicuramente si possono ottenere risultati soddisfacenti con uno sforzo decisamente minimo. Infine è da sottolineare l'integrazione con l'ambiente di Visual Studio al quale certamente siamo già abituati. Infine dobbiamno dire che si tratta di un prodotto scalabile utilizzabile sia in grandi cotesti sia in piccole e medie produzioni

Carmelo Scuderi

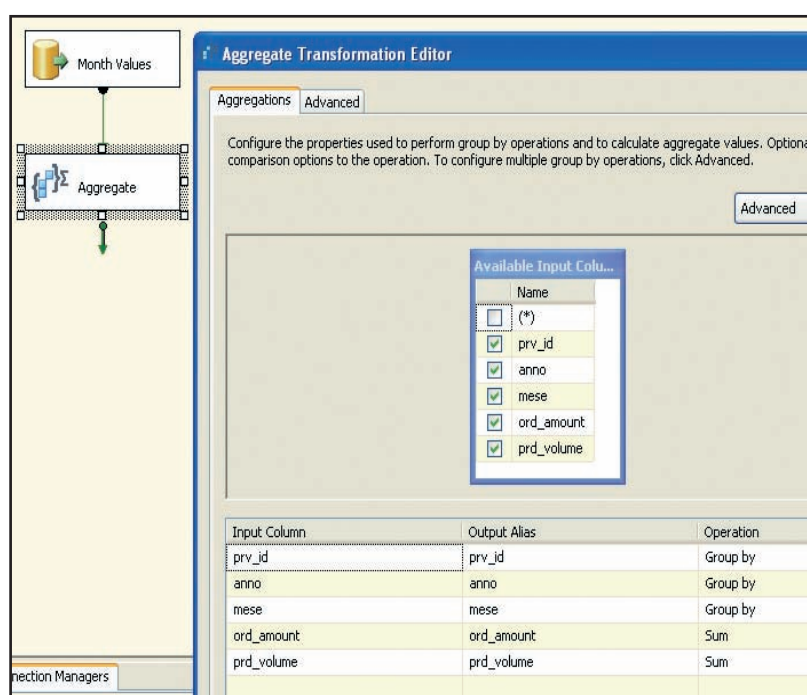



Fig. 10: *Aggreghiamo i dati*

UN LINGUAGGIO TANTI DEVICE

CELLULARI, PALMARI, TABLET PC, WEB BROWSER, E ALTRE DECINE DI PERIFERICHE AFFOLLANO ORMAI IL MERCATO DELL'INFORMATICA. MA COME SVILUPPARE APPLICAZIONI IN GRADO DI ESSERE VISUALIZZATE CORRETTAMENTE OVUNQUE?

WPF/E sta per WPF Everywhere, e WPF sta per Windows Presentation Foundation, la tecnologia di presentazione introdotta con il recente lancio di .NET 3.0. WPF/E non è un nome definitivo, ma un nome in codice così come lo era WinFX prima di trasformarsi in .NET 3.0.
WPF/E porta WPF sul web, e su diverse piattaforme hardware e software, inclusi i sistemi Macintosh. Esso consente la creazione di interfacce grafiche avanzate, sia nell'aspetto che nella possibilità di interazione dell'utente. Come in WPF, anche in questo caso è XAML (eXtensible Application Markup Language) che la fa da padrone, il linguaggio a markup per la progettazione dell'interfaccia grafica. WPFE permette ad un browser di renderizzare codice XAML allo stesso modo in cui fa con HTML, naturalmente XAML per WPFE è un sottoinsieme si quello dedicato alle applicazioni desktop.

ARCHITETTURA DI .WPF/E

.NET 3.0 è costruito al di sopra di .NET 2.0. La figura 1 mostra una vista di insieme della nuova architettura.
I componenti che costituiscono .NET Framework 3.0, si pongono al di sopra e funzionano in collaborazione con il .NET framework 2.0. Ciò significa fra l'altro che le "vecchie" applicazioni scritte per .NET 2.0 continueranno a funzionare senza alcuna modifica anche dopo aver installato il nuovo framework, ciò perché il CLR non ha subito alcuna modifica, così come sono rimaste invariate le librerie della Base Class Library. Nessuna paura di incompatibilità quindi.
WPF/E è un sottoinsieme di WPF, progettato per funzionare sul Web e quindi su diverse piattaforme. Ideale per gestire anche i device mobili quindi.

CREARE UN PROGETTO WPF/E

Dopo aver installato l'SDK di WPF/E (esattamente la CTP di dicembre 2006 al momento in cui scrivo), è possibile installare il template per Visual Studio 2005, basta andare in start->programmi->WPFE SDK->Tools e quindi lanciare l'installazione Install WPFE VS Template.
È necessario sottolineare che è richiesto anche il Web Application Project di Visual Studio 2005. A questo punto dall'IDE basta creare un nuovo progetto e scegliere come tipologia proprio WPFE.

CREARE CONTROLLI WPF/E

Dopo l'installazione dei componenti runtime di WPF/E, il browser Internet potrà utilizzare il plugin apposito per la creazione di uno o più controlli WPFE in una pagina HTML. Un controllo WPFE è un oggetto all'interno del quale potrà essere mostrato ed eseguito il contenuto WPFE stesso, cioè XAML. Ogni browser ha naturalmente il proprio modo per creare un simile oggetto. Internet Explorer su Windows utilizza per esempio un controllo ActiveX, mentre le altre piattaforme utilizzano la tecnologia a plug-in di Netscape. Di conseguenza su Internet Explorer si utilizzerà un tag Object all'interno dell'html, mentre per esempio su Firefox sarà necessario utilizzare un tag Embed.
Ovvero, dopo avere creato un file xaml1.xaml, esso potrà essere mostrato in un controllo WPF/E su Internet Explorer in questo modo:

```
<object
  id="WpfeControl"
  width="300"
  height="300"
  classid="CLSID:32C73088-76AE-40F7-AC40-
```

```
                                81F62CB2C1DA">

   <param name="BackgroundColor"
                              value="#ffebcd" />
   <param name="SourceElement" value=null />
   <param name="Source" value="xaml1.xaml" />
   <param name="WindowlessMode" value="true"
                                              />
   <param name="MaxFrameRate" value="30" />
   <param name="OnError"
                         value="myErrorHandler" />
</object>
```

Tale codice non funzionerà però su Firefox. Per creare pagine contenenti controllo WPF/E visualizzabili su qualunque piattaforma e browser, l'approccio migliore è l'utilizzo del file aghost.js, contenuto nell'SDK di WPF/E e quindi praticamente l'approccio ufficiale per sviluppare applicazioni con tale nuova tecnologia.
Tale file non è altro che un helper javascript, che si occupa semplicemente di verificare la piattaforma su cui sta eseguendo, e quindi generare un tag object oppure un tag embed. Una volta copiato il file aghost.js nella directory del file html che dovrà contenere il controllo WPFE, basterà inserire all'interno di quest'ultimo il seguente codice, praticamente standard, che con lievi modifiche, basta modificare il percorso del file XAML, potrà essere copiato e incollato dove necessita:

```
<html>
  <head>
    <title>Esempio WPFE</title>
    <script type="text/javascript"
                           src="aghost.js"></script>
  </head>
  <body>
       <div id="agControlHost">
       </div>

       <script type="text/javascript">
              // Crea il controllo
              //Active/X per  WPF/E.
              new agHost(
                          "agControlHost",
// l'elemento HTML che ospiterà il controllo WPF/E
"agControl1",
// The ID of the WPF/E ActiveX control to create.
                          "300px",
// larghezza del controllo.
                         "300px",
// altezza del controllo.
                          "#F6D606",
// colore di sfondo del controllo.
                          null,
```

```
// SourceElement
              (name del tag script contenente XAML)
                            "xaml1.xaml",
           // l'uri del file XAML
                            "false",
          // IsWindowless
                            "30",
          // MaxFrameRate
                            null
          // OnError, nome
                      del metodo gestore di errori.
              );
           // Crea una variabile globale per il
          // controllo WPF/E,
              var agControl =
          document.getElementById("agControl1");
       </script>
  </body>
</html>
```

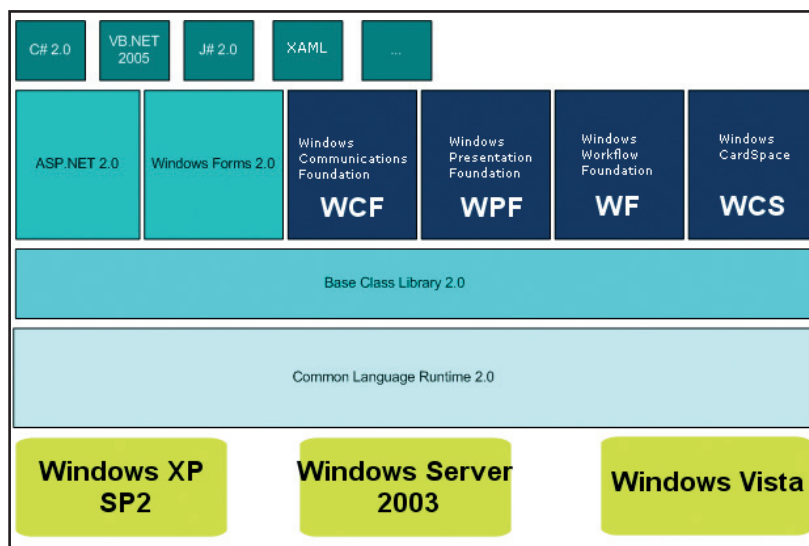



Figura 1: Architettura di .NET 3.0

INSTALLAZIONE DI .WPF/E

Sul WPF/E development center del sito microsoft, sono disponibili diversi download. Per visualizzare pagine web che utilizzano la tecnologia è necessario innanzitutto scaricare ed installare il WPFE Runtime (o WPFE Plugin), disponibile in versione CTP (Community Technology Preview) per Windows e Macintosh. Se l'installazione è andata a buon fine potrete provare a visualizzare qualche esempio, ormai abbondanti in rete, per esempio l'intestazione del sito http://thewpfblog.com è fatta in WPF/E e sullo stesso sito sono presenti diversi esempi, anche corredati di sorgente.
Per sviluppare e scrivere codice è invece necessario il WPF/E SDK, insieme al quale è naturalmente raccomandato Visual Studio 2005, in quanto l'SDK installerà anche i template per creare progetti WPF/E.
Da notare, che al momento della stesura dell'articolo, l'installazione del template non è compatibile con il Service Pack 1 di Visual Studio 2005.

Nel precedente codice HTML si noti il percorso del file XAML da utilizzare, che in questo caso verrà cercato nella stessa cartella, xaml1.xaml. Creiamo ora per esempio un ellisse ed una etichetta di testo in questa maniera:

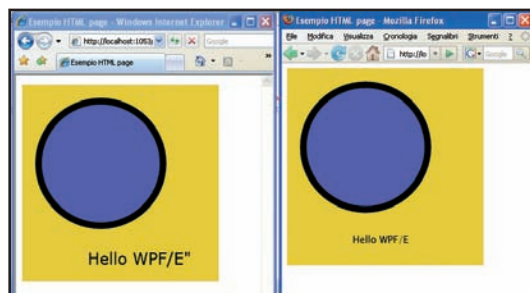



Figura 2: Esempio WPFE in IExplorer e Firefox

Basta ora salvare il codice xaml nel file xaml1.xaml ed aprire la pagina HTML nel proprio browser (**figura 2**).
È possibile anche inserire direttamente il contenuto XAML in un elemento script, naturalmente non è il caso farlo, se il codice xaml è lungo e complesso.

NOTA

REFERENZIARE DLL NEI PROGETTI

In Visual Studio per referenziare un assembly in un progetto è sufficiente cliccare con il tasto destro del mouse sul nome del progetto, scegliere la voce Add Reference... , cliccare sul tab Browse e selezionare dal disco rigido la DLL che s'intende referenziare nel progetto.

CENNI DI XAML

XAML è un linguaggio dichiarativo con il quale è possibile creare degli elementi di interfaccia grafica, in maniera analoga ad un qualunque altro linguaggio di markup cme HTML o XML. Per rispondere invece agli eventi o manipolare da codice gli elementi e gli oggetti dichiarati in XAML è possibile utilizzare JavaScript.
I file XAML sono in sintesi dei file in formato XML con estensione .xaml extension. Il seguente esempio o quelli visti in precedenza nell'articolo, mostrano un possibile frammento di XAML.

```
<Canvas
xmlns="http://schemas.microsoft.com/client/2007
"
xmlns:x="http://schemas.microsoft.com/winfx/20
                                    06/xaml">
  <Ellipse
    Width="100"
    Height="100"
    Fill="Red" />
</Canvas>
```

Ci sono diversi modi per dichiarare oggetti ed impostarne le proprietà in XAML. Come un qualsiasi elemento XML si utilizzano tag di apertura e di chiusura.

```
<oggetto>
</oggetto>
```

Nell'esempio precedente "oggetto" è il nome dell'oggetto che si vuole istanziare. Per esempio, per creare un oggetto Canvas in XAML si scriverà, come già visto:

```
<Canvas>
</Canvas>
```

Un oggetto *Canvas* è l'oggetto contenitore principale in WPFE. Esso conterrà e posizionerà tutti gli altri oggetti che si vorranno visualizzare all'interno di un controllo WPFE. Per aggiungere un oggetto ad un Canvas basta inserirlo fra i tag Canvas. L'esempio seguente aggiunge un ellisse ad un Canvas:

```
<Canvas>
        <Ellipse>
        </Ellipse>
</Canvas>
```

Naturalmente ogni oggetto avrà diversi attributi e proprietà, che possono essere impostati in due diversi modi, il primo è la sintassi ad attributi. Per esempio se si vuole impostare la dimensione dell'ellisse, cioè la sua larghezza e la sua altezza al valore di 100 pixel, ed il colore di riempimento al valore Red, si può scrivere:

```
<Canvas>
        <Ellipse Width="100" Height="100"
                                  Fill="Red">
        </Ellipse>
</Canvas>
```

La seconda possibilità è quella di utilizzare la Property Element Syntax. In questa seconda maniera ogni proprietà deve corrispondere ad un elemento XML, così come mostrato di seguito:

DEBUG DI JAVASCRIPT

Sviluppando con WPFE, ma non solo, può essere utile eseguire il debug del codice javascript presente nella pagina HTML direttamente da Visual Studio ed utilizzandone le potenzialità, per esempio con la possibilità di definire breakpoint e di analizzare il valore delle variabili e degli oggetti. Per abilitate il debug del codice javascript è necessario, aprire la schermata delle Opzioni Internet dal dal menù Strumenti di Internet Explorer, e nella tab Avanzate deselezionare la voce Disable Script Debugging.

```
<Canvas>
        <Ellipse Width="100" Height="100">
<Ellipse.Fill>
        </SolidColorBrush>
</Ellipse.Fill>
        </Ellipse>
</Canvas>
```

Ogni proprietà può supportare entrambe le modalità di impostazione, e la scelta dipende dal tipo di oggetto che la proprietà accetta. Se la proprietà è di un tipo primitivo, per esempio interi o stringhe, può essere utilizzata solo la sintassi ad attributi.
Infine notiamo che è possibile creare dei Canvas innestati, per esempio così:

```
<Canvas
xmlns="http://schemas.microsoft.com/client/2007
                                               "
xmlns:x="http://schemas.microsoft.com/winfx/20
                                       06/xaml">

  <Canvas Height="100" Width="100"
              Canvas.Left="30" Canvas.Top="30"
                          Background="Red">
    <TextBlock Text="Hello Canvas 1"
                          FontFamily="Verdana"
                                FontSize="14">
    </TextBlock>
  </Canvas>

  <Canvas Height="100" Width="100"
             Canvas.Left="230" Canvas.Top="30"
                        Background="Yellow">
    <TextBlock Text="Hello Canvas 2"
                          FontFamily="Verdana"
                   FontSize="14"></TextBlock>
  </Canvas>

</Canvas>
```

Il Canvas principale, contiene a sua volta due canvas, ed ognuno dei due contiene un TextBlock.

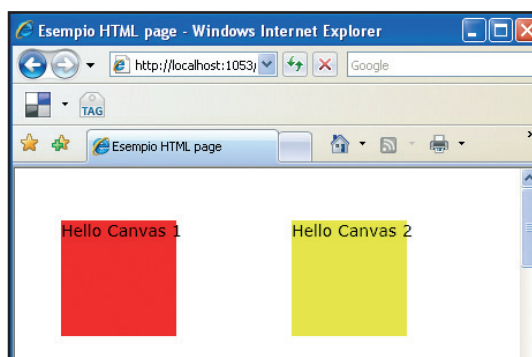



Figura 3: Canvas di WPFE innestati

FORME E DISEGNI IN WPF/E

Ora che abbiamo dato uno sguardo di insieme a XAML e WPFE, si vedrà come realizzare, in XAML appunto, diverse forme da visualizzare in un controllo WPFE ospitato in una pagina Web
WPF/E fornisce tre forme di base: Ellipse, Rectangle, e Line. Abbiamo già incontrato i primi due, e sappiamo come creare quindi una forma ovale o un cerchio mediante l'elemento Ellipse, impostando il diametro verticale e orizzontale con le proprietà Width ed Height property. Analogamente Rectangle permette di creare un quadrato od un rettangolo, opzionalmente con gli angoli arrotondati, utilizzando le proprietà RadiusX e RadiusY per controllare la curvatura.
Per le linee, invece di utilizzare le proprietà Width e Height, si controlla la lunghezza e la posizione impostando le proprietà X1,Y1, X2, e Y2, vale a dire le coordinate del punto di partenza e del punto in cui finisce la linea.
L'esempio seguente disegna un ellisse, un rettangolo, ed una linea in un canvas.

```
<Canvas

xmlns="http://schemas.microsoft.com/client/
                                        2007"

xmlns:x="http://schemas.microsoft.com/winfx/20
                                       06/xaml">
  <Rectangle Height="220" Width="220"
                 Canvas.Left="60" Canvas.Top="60"
      Stroke="Green" StrokeThickness="10">
    <Rectangle.Fill>
      <LinearGradientBrush StartPoint="0,0"
                                  EndPoint="1,1">
        <GradientStop Color="Yellow" Offset="0.0"
                                              />
        <GradientStop Color="Red" Offset="0.75"
                                              />
      </LinearGradientBrush>
    </Rectangle.Fill>
  </Rectangle>
  <Ellipse Height="120" Width="120"
                  Canvas.Left="5" Canvas.Top="5"
      Stroke="Black" StrokeThickness="10">
    <Ellipse.Fill>
      <RadialGradientBrush>
        <GradientStop Color="Yellow" Offset="0.0"
                                              />
        <GradientStop Color="Red" Offset="0.5" />
        <GradientStop Color="LimeGreen"
                                  Offset="1.0" />
      </RadialGradientBrush>
    </Ellipse.Fill>
```

BIBLIOGRAFIA E SITOGRAFIA

WPFE Development Center, http://www.msdn.com/wpfe

Esempi e tutorial su WPF e WPFE, http://www.thewpfblog.com/

NOTA

VISUAL STUDIO E NAMESPACE

Il namespace di WPF/e, http://schemas.microsoft.com/client/2007, non è riconosciuto da Visual Studio, perciò non funzionerà. Per risolvere il problema bisogna copiare il file wfpe.xsd dalla cartella dell'SDK a quella degli schemi di VS e cioè da C:\Programmi\Microsoft SDKs\WPFE\Help\XSD a C:\Programmi\Microsoft Visual Studio 8\Xml\Schemas

```
  </Ellipse>
  <Line X1="200" Y1="10" X2="20" Y2="300"
      Stroke="Blue" StrokeThickness="3"/>
</Canvas>
```

La figura 3 mostra il risultato.

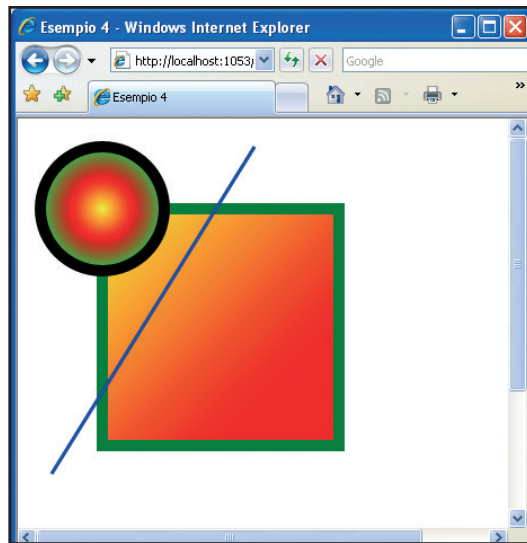



Figura 4: disegno di forme con xaml e wpfe

ANIMAZIONI IN XAML

Fino adesso abbiamo utilizzato XAML per creare dei semplici disegni o delle semplici forme. XAML e WPFE permettono di creare anche animazioni. Il primo passo è quello di creare un oggetto da muovere, cioè da animare. Tale oggetto, in questo caso, dovrà avere anche un nome, in maniera da poterlo referenziare per creare il movimento.
Per il nostro primo esempio utilizzeremo un nuovo oggetto, vale a dire Image, così da approfittarne per introdurre anche il suo utilizzo in wpfe.
Come abbiamo già visto, tutto deve essere aggiunto ad un Canvas. Se dunque abbiamo già un file esistente, in questo caso pallone.jpg, per aggiungerlo al Canvas basterà creare l'oggetto Image in questa maniera:

```
<Canvas>
  <Image x:Name="pallone" Source="pallone.jpg"
                          Canvas.Left="10" />
</Canvas>
```

A questo punto abbiamo un oggetto da animare. Per ottenere ciò è necessario un EventTrigger che esegua un'azione quando qualcosa lo richieda. Questo qualcosa è appunto un evento, il quale sarà specificato dalla proprietà RoutedEvent. Nel nostro esempio vogliamo che l'EventTrigger faccia partire l'animazione e quindi si utilizzerà come azione BeginStoryboard, mentre l'istante di inizio sarà determinato dall'evento Loaded dell'oggetto Canvas.

```
<Canvas>
  <Canvas.Triggers>
    <EventTrigger RoutedEvent="Canvas.Loaded">
      <EventTrigger.Actions>
        <BeginStoryboard>
          <Storyboard>
          ...
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger.Actions>
    </EventTrigger>
  </Canvas.Triggers>
  <Image x:Name="pallone" Source="pallone.jpg"
                          Canvas.Left="10" />
</Canvas>
```

Ma cosa deve essere animato e come? Ciò viene definito dallo Storyboard. Vediamo come crearne uno per spostare il nostro pallone da una posizione ad un'altra del Canvas. In WPF/E ogni animazione viene creata modificando una o più proprietà dell'oggetto, nell'esempio modificheremo la posizione verticale dell'immagine e quindi la proprietà Top. Dato che la proprietà è di tipo double è necessario utilizzare una DoubleAnimation, identificare l'oggetto obiettivo e la proprietà obiettivo, il valore a cui impostarla e la durata.

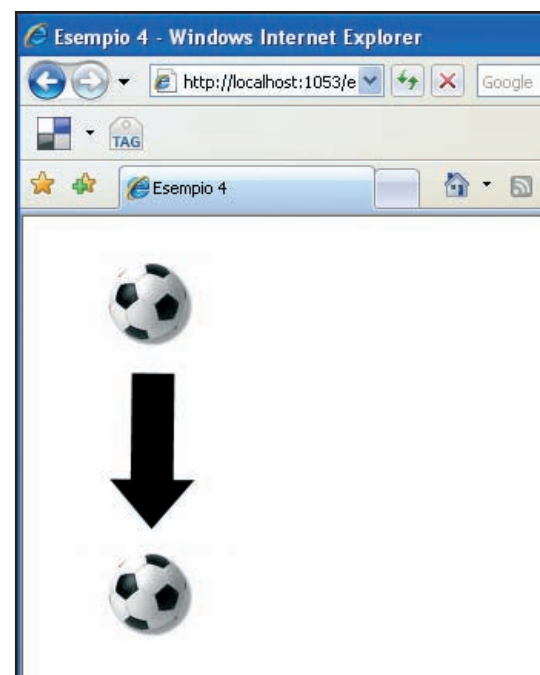



Figura 5: Non si nota dalla figura ma il pallone cade!

```
<DoubleAnimation
    Storyboard.TargetName="pallone"
    Storyboard.TargetProperty="(Canvas.Top)"
    To="200" Duration="0:0:1" />
```

La cosa forse più affascinante è che il tutto viene impostato semplicemente per mezzo di un linguaggio di markup.
Naturalmente è possibile creare anche altri tipi di animazione, per esempio per variare il colore di un oggetto, la dimensione, e così via.

PROGRAMMAZIONE JAVASCRIPT

WPF/E consente di eseguire del codice javascript al verificarsi di un evento, per esempio quando viene cliccato il pulsante del mouse. Basta definire un gestore dell'evento così come indicato di seguito.
Nel file XAML, bisogna aggiungere un attributo all'oggetto che utilizzerà il gestore secondo la seguente sintassi.

```
nomeEve0nto=”javascript:nomeFunzione”
```

nomeEvento è naturalmente l'evento da gestire mentre nomeFunzione è il nome della funzione che gestirà l'evento stesso.
Definiamo nel prossimo esempio di xaml la gestione degli eventi MouseLEftButtonDown del Canvas, ed inoltre MouseEnter e MouseLeave di un ellisse.

```
<Canvas Height="300" Width="300"
xmlns="http://schemas.microsoft.com
                                /client/2007"
xmlns:x="http://schemas.microsoft.com/winfx/20
                                      06/xaml"
   Background="transparent"
                       MouseLeftButtonDown=
                  "javascript:createRectangle">

<Ellipse x:Name="e1"
              MouseEnter="javascript:ellipseEnter"
MouseLeave="javascript:ellipseLeave"
             Height="100" Width="100"
           Canvas.Left="80" Canvas.Top="30"
             Stroke="Black"
        StrokeThickness="5" Fill="LightBlue"/>
</Canvas>
```

Nel file html, o in un file javascript separato da referenziare mediante un attributo src, inseriamo a questo punto le funzioni di gestione degli eventi definite nel codice XAML:

```
<script type="text/javascript">
     function ellipseEnter(sender, args) {
         sender.stroke = "red";
     }
     function ellipseLeave(sender, args) {
         sender.stroke = "black";
     }

     function ellipseMouseDown(sender, args) {
         sender.fill = "yellow";
     }
     function createRectangle(sender, args) {
         var agControl =
         document.getElementById("agControl1");
var e = agControl.createFromXaml('<Rectangle
                      Canvas.Left="'+args.x+'"
Canvas.Top="'+args.y+'" Height="20" Width="20"
                               Fill="Blue"/>');
         var canvas = sender;
         canvas.children.Add(e);
     }
  </script>
```

Ad ogni clic sul Canvas verrà creato un nuovo rettangolo, grazie al metodo CreateFromXaml del controllo Wpfe, che prende in ingresso una stringa contenente codice XAML.
Invece spostando il mouse sull'ellisse esso cambierà colore del bordo, riportandolo al colore nero uscendo nuovamente da esso, creando il classico effetto MouseOver.

CONCLUSIONI

Nell'articolo si è data un'introduzione della tecnologia WPF/E di Microsoft che permetterà di creare applicazioni web graficamente accattivanti, utilizzando codice XAML praticamente identico a quello che la farà da padrone nel prossimo futuro per quanto riguarda le applicazioni desktop. WPF/E è ancora in uno stadio embrionale ma promette veramente bene. C'è da dire che il Web paradossalmente è sufficientemente lento a recepire innovazioni tecnologiche di tale rilievo. Attualmente la stragrande maggioranza del web gira ancora con modalità del tutto standard utilizzando al più tecniche di connessione ai database per la gestione dei dati. L'arrivo di XAML e di WPE tuttavia apre scenari innovativi, come sempre chi per primo inizierà la sperimentazione su queste tecniche molto probabilmente sarà anche colui che si troverà avvantaggiato in futuro. Non abbia e dunque paura di innovare.

Antonio Pelleriti

L'AUTORE

Antonio Pelleriti, ingegnere informatico, sviluppa software da più di dieci anni e si occupa di .NET sin dalla prima versione Beta. È content manager della .NET community www.dotnetarchitects.it. Può essere contattato per suggerimenti, critiche o chiarimenti all'indirizzo e-mail antonio.pelleriti@dotnetarchitects.it

ARCHITETTURE MODULARI

COME CREARE SOFTWARE ANCHE COMPLESSO SCEGLIENDO IL LINGUAGGIO PIÙ ADATTO PER OGNI MODULO. IN QUEST'ARTICOLO SVILUPPEREMO UN'APPLICAZIONE CHE RIVELA EVENTUALI FILE CORROTTI O DANNEGGIATI NEL VOSTRO HARD DISK

Nel precedente numero di ioProgrammo abbiamo indagato su come fosse possibile riutilizzare del codice scritto in C/C++ nella piattaforma .NET. Come caso di studio abbiamo affrontato gli algoritmi di hashing, in particolare abbiamo ripreso una libreria per il calcolo dell'estratto CRC32 e l'abbiamo integrata in una GUI scritta in Visual Basic .NET.
Anche il seguente articolo sarà all'insegna del "multi-linguaggio", proprio per diffondere la cultura secondo la quale il vero programmatore professionista non è il super specialista di un linguaggio, bensì è colui che riesce a progettare delle architetture software complesse in maniera astratta e poi sa implementare le varie parti nel linguaggio più consono per quel contesto. Questa mentalità vi permetterà, tra le tante altre cose, di "rivendervi" nel competitivo mercato dello sviluppo del software nei campi più disparati. Dopo questa piccola breve premessa che mi sono permesso di fare per illustrarvi gli "intenti" di quest'articolo e del precedente riprendiamo alcuni concetti affrontati la volta scorsa.

ALGORITMI HASH

Per prima cosa ribadiamo cosa siano gli algoritmi di hash e quali siano le loro proprietà più interessanti. Come detto la volta scorsa essi sono una famiglia di algoritmi che soddisfa questi requisiti:

1) L'algoritmo ritorna una stringa di lunghezza fissa di bit a partire da qualsiasi lunghezza di byte in ingresso. Tale stringa è detta Digest.

2) La stringa è univoca per ogni sequenza di byte e ne è un identificatore. È per questo che l'algoritmo è utilizzabile per la firma digitale.

3) L'algoritmo non è invertibile, ossia non è possibile ricostruire il documento originale (ingresso) a partire dalla stringa che viene ritornata in output. Un po' come succede per le impronte digitali: un'impronta identifica una persona ma da sola non ci dice nulla su come essa sia fatta.
Il bello di questi algoritmi è quindi la possibilità di avere digest diversi anche cambiando un solo bit dell'input. Gli immediati utilizzi (ma non sono gli unici) che possono venire in mente sono:

1) Poiché ogni digest è un identificatore univoco dell'ingresso è possibile trovare un file, non secondo il suo nome, bensì secondo il suo contenuto. Cerchiamo di chiarire subito questo concetto:

supponiamo di scrivere un semplice programmino in python di una singola riga di codice:

```
print "hello word"
```

e salviamolo come hello.py in formato UTF-8. Visto che si tratta del solito primo programma ci sarà una buona probabilità che, sparso per il nostro disco fisso, ci sia un file esattamente identico ma con un altro nome, supponiamo si chiami hello2.py. Ora potremmo pensare di fare ciò: calcolare l'estratto hash di hello.py e cercare tutti quelli che abbiano un digest identico a quello precedentemente calcolato.

2) Sempre per la proprietà di univocità ipotizziamo uno scenario diverso:

vogliamo scaricare da internet un file di grosse dimensioni (ad esempio un'immagine di una distribuzione linux). Sappiamo che la probabilità che tale file si corrompa durante il download aumenta con la dimensione del file stesso. In questi casi viene solitamente fornito assieme al file il suo digest. In questo modo sarà quindi possibile calcolare l'estratto hash del file scaricato e confrontarlo con quello fornito. Se i due valori coincidono, tutto è andato per il meglio, altrimenti significa che durante il download il file è stato corrotto.
Questi due concetti sono largamente utilizzati da un popolarissimo programma di peer to

peer come eMule. Infatti per stabilire da quante e quali fonti è possibile scaricare un file non si utilizza certo il nome, bensì proprio tramite l'estratto hash. Inoltre, una volta terminato il download, verifica la correttezza di quanto scaricato ricalcolando tale digest, proprio come illustrato nel punto 2.

CACCIA ALL'INTRUSO

Ora che abbiamo fatto una breve panoramica degli scenari d'impiego di questo tipo di algoritmi, entriamo più nello specifico dell'applicazione che vogliamo implementare.
I primi virus informatici che comparvero sulla scena si basavano su un principio molto semplice (almeno a parole e nella versione iper-semplificata che andrò ad esporvi). Essi non erano altro che programmi "maligni" che iniettavano il proprio codice sui file eseguibili e li utilizzavano come "ospiti" per essere eseguiti. Allo stesso tempo quando si seguiva un file "infetto", il virus aveva anche la possibilità di replicarsi su altri file eseguibili; in altre parole era il suo modo di replicarsi. In questo periodo, i primi antivirus sfruttavano proprio un meccanismo simile a quello descritto in precedenza: nel momento in cui si effettuava uno scan del proprio hard disk, l'antivirus manteneva un log dei digest dei file eseguibili incontrati, cosicché nella eventualità che un file fosse stato infettato appariva un warning riguardo il fatto che un file è stato modificato (quelli che utilizzano il PC dai tempi del DOS sicuramente ricorderanno questo tipo di messaggi di alert).
Naturalmente ormai sia i virus che gli antivirus usano tecniche ben più sofisticate di quelle appena descritte e non è nostra intenzione metterci a sviluppare un antivirus completo, bensì vogliamo tenere traccia delle modifiche apportate ai del nostro hard disk e se per caso troviamo un file eseguibile modificato... beh allora ci può essere qualcosa che non va! Questa nostra applicazione ci permetterà di approfondire vari aspetti:

1. Sviluppare un'architettura i cui moduli sono scritti in diversi linguaggi
2. Aggiungere dei controlli GUI custom nell'IDE Visual Studio 2005 o 2003
3. Utilizzare l'XML con XQuery complesse
4. Utilizzare le classi di .NET nel namespace System.XML

Come abbiamo anticipato all'inizio, saper programmare significa saper scrivere del buon codice nel linguaggio giusto. La figura 1 mostra i vari layers della nostra applicazione. L'ordine con cui sono stati disegnati non è casuale: infatti muovendoci dal basso verso l'alto saliamo di livello anche per quel che riguarda il modello di programmazione. I primi due strati sono stati già esposti e sviluppati nell'articolo precedente e riguardano l'implementazione dell'algoritmo CRC32 in C++ nativo e la sua integrazione in .NET tramite la classe wrapper.
È stato naturale realizzare l'implementazione di un algoritmo pesante dal punto di vista computazionale come quello del CRC32 in C++, proprio per la sua efficienza. A questo punto ci occuperemo degli ultimi due strati: rispettivamente quello in C# e quello in Visual Basic .NET.

Figura 1: I layers di cui è composta la nostra applicazione

SCRIVERE UN BUON XML

È ormai giunto finalmente il momento di addentrarci nel codice. Cominciamo con quello sviluppato in C#. Come si può notare dalla figura 1, tale layer implementerà un log. Più precisamente il nostro scopo è quello di organizzare una struttura XML che contenga tutte le informazioni che otteniamo quando la nostra applicazione effettua uno scan dei file. Inoltre vorremo anche una classe che fornisca la logica (ossia un set di metodi) per poter confrontare i valori hash precedentemente ottenuti con quelli attuali. Partiamo quindi da come sarà strutturato il nostro file XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<Log poly="" defaultDate="">
  <list date="">
    <item file="" hash=""/>
    ............
  </list>
  ....................
  <list date="">
     ............
```

```
  </list>
</Log>
```

Per tradurre con “parole umane” quanto riportato in maniera informale nel codice sopra (senza passare per un XML Schema che ci causerebbe solo complicazioni ulteriori) possiamo definire i seguenti punti:

1 – Il tag root è *Log* ed ha due attributi: *poly* e *defaultDate* che rispettivamente indicano il polinomio per la generazione del digest CRC32 (vedi articolo precedente) e la date che indica quale nodo list è quello di default (torneremo tra poco su questo concetto).
2 – Il tag di root può avere N nodi figli con tag *list*. Tali nodi conterranno tutte le informazioni ottenute ogni volta che si effettua un nuovo scan. Un nodo è distinguibile dall'altro per mezzo del suo attributo *date*. A questo punto possiamo anche comprendere meglio il significato dell'attributo *defaultDate*: esso indica quale tra questi nodi è quello preso per effettuare eventuali confronti con scansioni successive.
3 – Ogni nodo list può avere N nodi figli con tag *item*. Ognuno di questi rappresenta un singolo file analizzato: in particolare gli attributi *file* , *hash* rappresentano rispettivamente il nome completo del file ed il suo estratto hash.

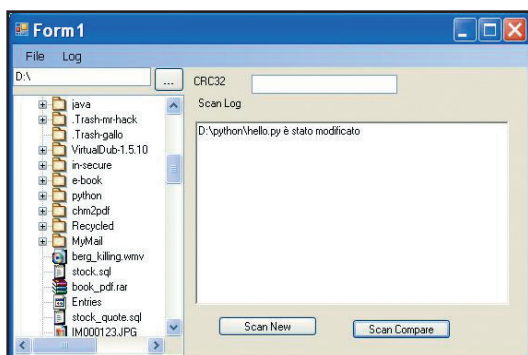



Figura 2: Modifica del file hello.py

La prima cosa che solitamente faccio quando voglio implementare un classe che interagisca a stretto contatto con una struttura XML è quella di fissarmi tutti i tag ed incapsulare un oggetto XmlDocument:

```
public class LogHash
 {
    private XmlDocument doc;
    private string currentPath;
    private const string logTag = "Log";
    private const string logDefaultDateAtt =
                                    "defaultDate";
    private const string logPolyAtt = "LogPoly";
    private const string listTag = "list";
    private const string listDateAtt = "date";

    private const string itemTag = "item";
    private const string itemFileAtt = "file";
    private const string itemHashAtt = "hash";
    ....
```

Un'altra cosa che di frequente torna utile è quella di fornire un metodo che permetta di ottenere un oggetto che istanzi tale classe a partire direttamente dal file XML stesso. Ritengo inoltre che sia maggiormente corretto definire tale metodo come statico perché è proprio compito di quest'ultimo fornire un'istanza della classe senza che sia necessario che se ne sia creata un'altra precedentemente.
Nel nostro caso tale metodo sarà semplicemente:

```
public static LogHash load(string logPath)
 {
    LogHash log = new LogHash();
    log.doc.Load(logPath);
    log.currentPath = logPath;
    return log;
 }
```

Va notato che il costruttore è stato dichiarato privato, questo per rendere ancora più esplicito il fatto che creare un oggetto da tale classe implica anche avere un side effect al di fuori dell'ambiente runtime.

```
private LogHash()
 {
    doc = new XmlDocument();
    currentPath = string.Empty;
 }
```

Questo aspetto diviene ancora più importante quando vogliamo creare un nuovo log. Infatti, per quanto scritto sopra, non ci possiamo limitare esclusivamente a creare un'istanza della classe, bensì dovremmo costruire anche la struttura del XML in questione e caricarlo sull'oggetto *doc*, in modo tale da avere già una struttura XML ben formata (well-formed).
Per fare ciò è stato implemento il metodo statico:

```
public static LogHash
             createNew(CRC32.EPolynomials poly)
 {
    LogHash log = new LogHash();
    StringBuilder sb = new StringBuilder();
    XmlWriter w = new XmlTextWriter(new
                       System.IO.StringWriter(sb));
    string now =
        DateTime.Now.ToLongDateString() +" " +
              DateTime.Now.ToLongTimeString();

    w.WriteStartElement(logTag);
    w.WriteAttributeString(logPolyAtt,
      Enum.GetName(typeof(CRC32.EPolynomials),
```

```
                                                  poly));
            w.WriteAttributeString(logDefaultDateAtt,
                                                   now);
            w.WriteStartElement(listTag);
            w.WriteAttributeString(listDateAtt, now);
            w.WriteEndElement();

            w.WriteEndElement();
            w.Close();
            log.doc.LoadXml(sb.ToString());
            return log;
        }
```

Figura 3: Output di una nuova scansione.

Ci sono molti modi per generare un XML con la piattaforma .NET ed in quest'articolo ne vedremo un paio. Il primo è quello riportato nel metodo sopra, ossia vedere un XML a basso livello, cioè come una sequenza di tag annidati da aprire e chiudere. Secondo quest'ottica quindi basta effettuare una concatenazione di stringhe e sappiamo che quando ci troviamo di fronte ad una simile esigenza è sicuramente conveniente usare lo StringBuilder. In particolare esso verrà utilizza dal writer come buffer di accumulazione per le stringhe che compongono il nostro XML, come si capisce dalla costruzione del writer stesso:

```
XmlWriter w = new XmlTextWriter(new
                        System.IO.StringWriter(sb));
```

Il codice restante non è altro che la creazione della struttura del XML mostrato in precedenza. L'unica cosa che penso sia utile sottolineare è:

```
w.WriteAttributeString(logPolyAtt,
        Enum.GetName(typeof(CRC32.EPolynomials),
                                          poly));
```

Il polinomio per la generazione del CRC32 è stato definito come un enumeration, allo stesso tempo abbiamo la necessità di serializzarlo su XML. Per ottenere ciò è stata scelta la via più "leggibile", ossia invece di salvare il valore numerico del polinomio stesso abbiamo salvato il nome dell'enumeration, che può assumere solamente due valori:

```
public enum EPolynomials
    {
        polyStandardReversed = -306674912,
        polyStandard = 79764919
    }
```

XML AD OGGETTI

Ora cerchiamo di capire l'altro modo di lavorare con un XML, ossia quello di utilizzare gli oggetti.
Caliamoci ancora una volta nella nostra applicazione. Ci rimangono due funzionalità importanti da dover implementare nella nostra classe di log: poter aggiungere una nuova lista per le scansioni successive alla prima e la possibilità di aggiungere nuovi tag item (cioè la coppia nome file – digest hash) ad una lista. Per realizzare ciò non abbiamo più utilizzato il writer che costruisce l'XML riga per riga, bensì abbiamo pensato all'XML come una serie di oggetti incapsulati uno dentro l'altro. Dato che la cosa è più facile a dirsi che a farsi riporto subito il codice in questione:

```
public void createNewList()
    {
        XmlElement newList =
                            doc.CreateElement(listTag);
        XmlAttribute newAtt =
                        doc.CreateAttribute(listDateAtt);
        string now =
            DateTime.Now.ToLongDateString() + " " +
                    DateTime.Now.ToLongTimeString();
        newAtt.Value = now;

doc.SelectSingleNode(logTag).Attributes[logDefaultDa
                                   teAtt].Value = now;
        newList.SetAttributeNode(newAtt);
        XmlNode root = doc.DocumentElement;
        root.AppendChild(newList);
    }

public void add(string file, uint hash)
    {

        XmlNode list = getListDefault();
        XmlElement item =
                        doc.CreateElement(itemTag);
        item.SetAttribute(itemFileAtt, file);
        item.SetAttribute(itemHashAtt, string.Empty
                                             + hash);
        list.AppendChild(item);
    }
```

Per ragioni di tempo non possiamo addentrarci nello specifico di ogni riga di codice, ciò che è importante comprendere è il diverso paradigma di

SCARICARE LE LIBRERIE

La libreria BaseCodeGeneratorWithSite, di cui parliamo nell'articolo è liberamente scaricabile all'indirizzo http://www.gotdotnet.com/Community/UserSamples/Details.aspx?SampleGuid=4AA14341-24D5-45AB-AB18-B72351D0371C

programmazione. Dato un oggetto *doc* che rappresenta tutto l'XML, tramite il metodo CreateElement creiamo un nuovo nodo appartenente al *doc*, che poi aggiungiamo nel punto che preferiamo. Un errore molto comune in questi casi è quello di creare un nuovo nodo XML semplicemente per mezzo dell'operatore new e poi tentare di chiamare il metodo append sul document. Se si operasse in questa maniera lo stesso metodo append lancerebbe un'eccezione poiché si sta tentando di aggiungere un nodo non appartenente al document.

CERCARE CON GLI XPATH

Prima di arrivare all'interfaccia grafica analizziamo un ultimo aspetto. Abbiamo sicuramente bisogno di cercare se un log contiene un file (per verificare se è stato già scansionato) e se contiene un determinato digest (per verificare se è stato corrotto). La soluzione più ingenua e computazionalmente onerosa potrebbe essere quella di scorrere tutti i nodi dell'XML all'interno di un ciclo for finché non troviamo il nodo con l'attributo del valore che cercavamo. In realtà XML fornisce una sintassi molto più potente per risolvere tale problema. Se vogliamo ad esempio il file C:\ReadMe.txt basterà impostare un Xpath come il seguente:

```
Log/list/item[@file='C:\ReadMe.txt']
```

Questa sintassi permette cioè di selezionare un nodo non solo in base alla sua posizione all'interno del documento XML, ma anche in base ai suoi valori. A questo punto il metodo sotto riportato non dovrebbe necessitare di ulteriori commenti:

```
public bool contain(uint hash)
 {
   XmlNode n = getListDefault();
   XmlNode item =
  n.SelectSingleNode(itemTag+"[@"+itemHashAtt+"
                         ='"+hash.ToString()+"']");
   return item != null;
 }
```

LA GUI

È giunta l'ora di cambiare nuovamente linguaggio di programmazione! Per costruire un'interfaccia grafica ho deciso di utilizzare ancora una volta il linguaggio che ritengo più idoneo e diretto per un compito del genere, ossia il Visual Basic.
Invece di perderci in mille discussioni su come dovrebbe essere strutturata tale GUI, diamo subito un'occhiata alla figura 2.
Sulla parte sinistra abbiamo un controllo grafico chiamato TreeView che ci permette di fare il browsing del nostra hard disk in maniera più che agevole (vedi BOX 1). Sopra questo controllo è presente una textbox, affiancata da un bottone, che permette di impostare quale sia la root da visualizzare nel treeView. In VB è veramente molto semplice catturare gli eventi e farne qualcosa all'interno della GUI. Nel nostro caso dobbiamo verificare che ogni volta che viene cambiato il testo nella textbox esso rappresenti un path valido ed in caso positivo cambiare la root di treeView (ad esempio in figura 2 sono passato da C:\ a D:\). Come potete vedere non c'è nulla di più semplice con il VB:

```
Private Sub TextBoxTree_TextChanged(ByVal
     sender As Object, ByVal e As System.EventArgs)
                    Handles TextBoxTree.TextChanged
   If IO.Directory.Exists(TextBoxTree.Text)
                                              Then
     treeView.Load(TextBoxTree.Text)
   End If
 End Sub
```

Grazie a Handles TextBoxTree.TextChanged tale metodo verrà invocato ogni volta che il testo all'interno della textbox viene cambiato.
Un'altra feature interessante è quella di visualizzare il digest ogni volta che selezionate un file, anche in questo caso basta scrivere:

```
Private Sub treeView_AfterSelect(ByVal sender As
                               Object, ByVal e As
           System.Windows.Forms.TreeViewEventArgs)
                      Handles treeView.AfterSelect
   If
       IO.File.Exists(treeView.SelectedNode.FullPath)
                                              Then

     TextBox1.Text =
   getCRC32(treeView.SelectedNode.FullPath).ToStrin
                                            g("x")
   End If
 End Sub
```

Anche in questo caso, alla luce di quanto detto per il metodo precedente, il codice è auto-esplicativo. Da notare che il metodo getCRC32 non fa altro che istanziare la classe scritta in C++ e calcolare il digest del file (di ciò ce ne siamo occupati nello scorso articolo).
Ora andiamo finalmente a realizzare ciò che ci eravamo prefissi all'inizio dell'articolo: effettuare delle scansioni per controllare quali file siano stati cambiati, o nella nostra ottica sarebbe meglio dire “corrotti”. Prima di tutto selezioniamo File -> New -> uno dei due polinomi, ciò farà sì che venga invocato il

metodo createNew visto precedentemente.
A questo punto selezioniamo la directory che desideriamo scansionare e premiamo il bottone Scan New. Il codice che sta dietro a tale bottone è il seguente:

```
Private Sub ButtonScan_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs)
                        Handles ButtonScan.Click
    Me.Cursor = Cursors.WaitCursor
    RichTextLog.Text = String.Empty
    Dim root As String =
                  treeView.SelectedNode.FullPath
    log.createNewList()
    scan(root)
    Me.Cursor = Cursors.Default

  End Sub

  Private Sub scan(ByVal path As String)
    If IO.File.Exists(path) Then
      Dim crcVal As UInteger = getCRC32(path)
      log.add(path, crcVal)
      Me.RichTextLog.Text &= path + "  " +
                  crcVal.ToString("X") + vbCrLf
    ElseIf IO.Directory.Exists(path) Then
      Dim info As New IO.DirectoryInfo(path)
      Dim file As IO.FileInfo
      For Each file In info.GetFiles
        scan(file.FullName)
      Next
      Dim subDir As IO.DirectoryInfo
      For Each subDir In info.GetDirectories
        scan(subDir.FullName)
      Next
    End If
  End Sub
```

Come avrete sicuramente notato è il metodo scan a “fare il lavoro sporco”, ossia scansionare tutti i file nella directory calcolarne il digest, stamparlo sulla GUI e richiamarsi ricorsivamente su tutte le sotto-directory. Un esempio di scansione è riportato in figura 3.
Un volta effettuata la scansione possiamo salvare i risultati con File -> Save As, per poi riaprirli in un secondo momento.
Come avrete di certo intuito dalla figura 3, nel mio esempio ho effettuato uno scan su una directory che conteneva due semplici programmini “hello word” scritti in python. All'inizio dell'articolo avevamo detto che basta cambiare un solo bit dello stream di input dell'algoritmo CRC32 per ottenere un digest diverso; così non ho fatto altro che aggiungere uno spazio alla fine del file hello2.py e cliccare poi sul bottone scan compare. Il codice di scan compare è riportato di seguito, mentre il risultato è visualizzabile in figura 2.

```
Private Sub compare(ByVal path As String)
    If IO.File.Exists(path) Then
      If Not log.contain(path) Then
        RichTextLog.Text &= path + " non
                              presente" + vbCrLf
      Else
        Dim crc As UInteger =
                                  getCRC32(path)
        If Not log.contain(crc) Then
          RichTextLog.Text &= path + " è
                               stato modificato"
        End If
      End If
    ElseIf IO.Directory.Exists(path) Then
      ................
      Next
    End If
  End Sub
```

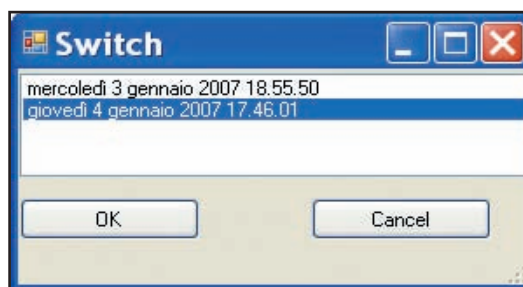



***Figura 4:** Passare da una scansione all'altra.*

Come vedete la logica è quella ricorsiva del metodo scan; in questo caso però si verifica se il file incontrato è già stato scansionato ed in caso positivo se il digest calcolato in precedenza è uguale a quello attuale. All'inizio avevamo deciso di strutturare l'XML in modo tale che prevedesse più liste che corrispondevano a scansioni diverse. È possibile passare da una scansione all'altra semplicemente andando su Log -> Cambia ed apparirà un dialogo come quello riportato in **figura 4**. Ad esempio rieffettuando una comparazione con una scansione successiva non viene più rilevata alcuna incongruenza.

CONCLUSIONI

In questi due ultimi articoli ho voluto principalmente invogliare chi legge a non “fossilizzarsi o iper-specializzarsi” nell'uso di un unico linguaggio, anche perché il nostro mondo è così dinamico che sarebbe veramente facile trovarsi spiazzati in poco tempo. D'altro canto se state leggendo questa rivista è ovvio che la vostra intenzione sia quella di rimanere sempre aggiornati!

Andrea Galeazzi

JAVA 6 E JAVASCRIPT COPPIA PERFETTA

IN QUESTO ARTICOLO PARLEREMO DI UNA POTENTE FEATURE DELLA VERSIONE 6 DEL LINGUAGGIO DELLA SUN. IN PARTICOLARE, APPRENDEREMO COME SIA POSSIBILE INVOCARE CODICE JAVASCRIPT DIRETTAMENTE DA JAVA E VICEVERSA

La versione 6 di Java Platform, Standard Edition (Java SE) include molte novità e miglioramenti. Le principali sono:

- Supporto per i linguaggi di scripting (lo vedremo nel presente articolo).
- Miglioramenti e nuove feature per quanto riguarda la sicurezza.
- Servizi Web integrati.
- Supporto per JDBC 4.0 (JSR 221).

Questa nuova versione del linguaggio della Sun non è da sottovalutare per l'importanza delle feature introdotte. Tra quelle citate, il supporto per i linguaggi di scripting è, a mio avviso, tra le più interessanti. Le specifiche dietro le quali si cela questa potente caratteristica di Mustang (nome in codice di Java 6) sono le JSR 223.

LE JSR 223

Le JSR 223 descrivono diversi meccanismi per permettere ai linguaggi di scripting di sfruttare le potenzialità di Java e viceversa. Sarà possibile, ad esempio, accedere direttamente all'oggetto XMLHttpRequest (cuore di Ajax) da applicazioni server scritte in Java.
Java 6 fornisce un motore di script basato su Rhino. Rhino è un'implementazione open source di JavaScript scritta interamente in Java.
Oltre a JavaScript è previsto il supporto per molti altri linguaggi di scripting attraverso l'implementazione delle JSR 223. Tra quelli disponibili figurano anche Python e Ruby, due linguaggi che stanno riscuotendo consensi sempre maggiori. Le corrispettive implementazioni JSR 223 sono, rispettivamente, Jython e JRuby.
Vedremo, ora, quali sono le API coinvolte in questo processo di integrazione dei linguaggi di scripting in Java e viceversa.

LE SCRIPTING API

Le API di cui stiamo parlando si trovano sotto il package javax.script. Questo package è composto da sei interfacce e sei classi illustrate in tabella 1.

Interfacce	Classi
Bindings	SimpleBindings
Compilable	CompiledScript
Invocable	ScriptException
ScriptContext	SimpleScriptContext
ScriptEngine	AbstractScriptEngine
ScriptEngineFactory	ScriptEngineManager

***Tabella 1:** Classi ed interfacce del package javax.script*

Le funzionalità offerte da queste interfacce e classi sono le seguenti:

- Esecuzione di script da Java. Questa feature permette di eseguire script espressi in uno dei linguaggi per i quali è disponibile un engine. Vedremo che, nativamente, Java 6 fornisce il supporto solo per JavaScript attraverso Rhino.
- Binding. Questa carattestica è opposta alla precedente, ossia tramite il binding è possibile accedere ad oggetti Java direttamente dagli script.
- Compilazione. Tale feature concerne il processo tramite il quale l'engine compila lo script in un codice intermedio. La cosa più interessante di questo processo è che tale compilazione viene "cachata", ossia le successive invocazioni sono più veloci poiché l'engine non dovrà ricompilare lo script.
- Invocazione. Tramite l'invocazione lo script viene invocato.
- Metadati sugli script engine. È possibile registrare script engine a run time ed avere informazioni utili (metadati) sui motori di script stessi.

GLI SCRIPT ENGINE

L'interfaccia che useremo più spesso, nel contesto delle scripting API, è ScriptEngine. È tramite l'engine, infatti, che interfacceremo Java con JavaScript. La classe che viene utilizzata per ottenere uno ScriptEngine è ScriptEngine Manager. Attraverso questa classe è anche possibile ottenere informazioni sui motori disponibili. Facciamo subito un esempio. Scriviamo il codice necessario per far apparire il classico "Hello World!" sulla console:

```
import javax.script.*;

ScriptEngineManager mgr = new
                    ScriptEngineManager();
ScriptEngine jsEngine =
            mgr.getEngineByName("JavaScript");
try
{
        jsEngine.eval("print('Hello World!')");
}
catch (ScriptException ex)
{
        ex.printStackTrace();
}
```

Per prima cosa importiamo il package riguardante le API in questione. In seguito, il codice visto, istanzia uno ScriptEngineManager. Poi, attraverso il metodo getEngineByName, ottiene un engine concernente il linguaggio JavaScript. Il metodo eval dell'interfaccia ScriptEngine ha la responsabilità di eseguire lo script passato come parametro. Tale metodo solleva un'eccezione di tipo ScriptException nel caso in cui la stringa passata come parametro non rappresenta uno script valido, ovvero se si verifica un errore durante l'esecuzione dello script.

Notate che nel codice precedente abbiamo utilizzato il metodo print. Questo metodo, per come è stato implementato in Rhino, visualizza sulla console il parametro passato come argomento. Ricordiamo che Rhino è un'implementazione open source di JavaScript, ovvero di ECMAScript che è lo standard su cui è basato JavaScript.

Oltre a getEngineByName, vi sono altri interessanti metodi esposti dalla classe ScriptEngineManager. Tra questi citiamo getEngineByExtension, getEngineByMimeType e getEngineFactories. I primi due possono essere utilizzati per ottenere uno script engine usando, rispettivamente, l'estensione tipica del linguaggio di script o il MIME type. Ad esempio, il seguente codice ottiene un motore per il linguaggio JavaScript (estensione js):

```
ScriptEngine jsEngine =
            mgr.getEngineByExtension("js");
```

Il metodo getEngineFactories, invece, restituisce una lista di ScriptEngineFactory. La sua signature è la seguente:

```
public java.util.List<ScriptEngineFactory>
                    getEngineFactories()
```

Metodo	Restituisce
public java.lang.String getEngineName()	Il nome completo dello script engine
public java.lang.String getEngineVersion()	La versione dello script engine
public java.lang.String getLanguageName()	Il nome del linguaggio di scripting supportato
public java.lang.String getLanguageVersion()	La versione del linguaggio di scripting supportato
public java.util.List<java.lang.String> getNames()	Una lista di stringhe che rappresentano i nomi dei motori di script supportati
public java.lang.Object getParameter(java.lang.String key)	Il valore dell'attributo specificato
public javax.script.ScriptEngine getScriptEngine()	Un'istanza dello script engine associato con questa factory

Tabella 2: Alcuni dei metodi esposti dall'interfaccia ScriptEngineFactory

L'INTERFACCIA SCRIPTENGINEFACTORY

L'interfaccia ScriptEngineFactory espone i metodi utili per ottenere metainformazioni sugli script engine. I metodi principali di quest'interfaccia sono illustrati in **tabella 2**.

Quello che segue è un esempio su Script EngineFactory:

```
ScriptEngineManager mgr = new
                    ScriptEngineManager();
List<ScriptEngineFactory> factories =
                    mgr.getEngineFactories();
for (ScriptEngineFactory factory : factories)
{
        System.out.println("Esempio su
                    ScriptEngineFactory");
        String engineName =
                    factory.getEngineName();
        String engineVersion =
                    factory.getEngineVersion();
        String languageName =
                    factory.getLanguageName();
        String languageVersion =
                    factory.getLanguageVersion();
        System.out.printf("\tScript Engine: %s
            (%s)\n", engineName, engineVersion);
    System.out.printf("\tLinguaggio: %s (%s)\n",
            languageName, languageVersion);
```

```
        List<String> engNames =
                        factory.getNames();
        for (String name : engNames)
        {
                System.out.printf("\tEngine Alias:
                                %s\n", name);
        }
}
```

Il precedente codice istanzia uno ScriptEngine Manager. Utilizza quest'oggetto per ottenere la lista delle factory disponibili. Ciclando sulle factory così ottenute, visualizza il nome e la versione dell'engine ed il nome e la versione del linguaggio supportato. In seguito ottiene una lista degli alias per questo engine chiamando il metodo getNames sulla factory. Ciclando su questa lista vengono visualizzati i vari alias. L'output ottenuto dall'esecuzione del precedente codice è il seguente:

```
Esempio su ScriptEngineFactory
        Script Engine: Mozilla Rhino (1.6 release 2)
        Linguaggio: ECMAScript (1.6)
        Engine Alias: js
        Engine Alias: rhino
        Engine Alias: JavaScript
        Engine Alias: javascript
        Engine Alias: ECMAScript
        Engine Alias: ecmascript
```

Come è possibile notare dall'output prodotto, Java 6 fornisce nativamente il supporto per il solo JavaScript. Come abbiamo già detto, tuttavia, vi sono numerose implementazioni open source per svariati altri linguaggi.

JAVASCRIPT DA CODICE JAVA

Vediamo, ora, di sfruttare JSR 223 per eseguire codice JavaScript da Java. Vi sono tre possibili scenari:

- Esecuzione di uno script rappresentato tramite una stringa.
- Esecuzione di uno script contenuto all'interno di un file.
- Compilazione ed invocazione di una funzione JavaScript definita dall'utente.

Il primo caso l'abbiamo già visto. Per comodità riportiamo l'esempio di seguito:

```
import javax.script.*;
ScriptEngineManager mgr = new
                        ScriptEngineManager();
ScriptEngine jsEngine =
```

```
                mgr.getEngineByName("JavaScript");
try
{
        jsEngine.eval("print('Hello World!')");
}
catch (ScriptException ex)
{
        ex.printStackTrace();
}
```

Come si può vedere, basta invocare il metodo eval dell'interfaccia ScriptEngine, passare la stringa che rappresenta lo script come parametro ed il gioco è fatto.
È anche possibile valutare uno script contenuto in un file; nel caso di JavaScript questo file ha, per convenzione, l'estensione .js. Vediamo un esempio:

```
ScriptEngineManager engineMgr = new
                        ScriptEngineManager();
ScriptEngine engine =
        engineMgr.getEngineByName("JavaScript");
InputStream is = null;
try
{
  is = new FileInputStream("./scripts/test.js");
  Reader reader = new InputStreamReader(is);
  engine.eval(reader);
}
catch (ScriptException ex)
{
  ex.printStackTrace();
}
catch (FileNotFoundException ex)
{
        ex.printStackTrace();
}
```

Il codice appena visto ottiene, tramite ScriptEngineManager, un'istanza di tipo ScriptEngine. Poi carica in un oggetto di tipo java.io.Reader il contenuto di un file JavaScript esterno. In seguito utilizza un overload del

NOTA

RISORSE UTILI

Key feature di Java 6:
http://java.sun.com/javase/6/features.jsp
Info su Rhino:
http://www.mozilla.org/rhino/
Scripting in Java:
https://scripting.dev.java.net
Java 6 SE API Specification:
http://java.sun.com/javase/6/docs/api/index.html
Java SE Downloads:
http://java.sun.com/javase/downloads/index.jsp
Articolo interessante su JSR 223:
http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/
Jython, implementazione 100% Java di Pyhton:
http://jython.sourceforge.net/Project/index.html
Jruby, implementazione 100% Java di Ruby:
http://jruby.codehaus.org/

VARARGS

Il metodo invokeFunction di Invocable ha la seguente firma:

```
public Object invokeFunction(String
                name, Object... args)
```

I tre puntini che seguono Object indicano che è possibile passare un numero variabile di argomenti. In pratica, è possibile passare questo numero variabile di argomenti o con il vecchio metodo che si usava in Java 1.4.x, ossia attraverso un array, oppure come una sequenza di argomenti separati da virgole. Ricordiamo che varargs, ossia la possibilità di passare un numero variabile di argomenti, è una caratteristica introdotta già dalla versione 1.5.0 di Java.

metodo eval per interpretare il contenuto del file in precedenza caricato. Come potete vedere, questa versione di eval si aspetta in ingresso un oggetto di tipo java.io.Reader.
Il file test.js, che si trova sotto la directory scripts, contiene il seguente codice:

```
function printMsg()
{
            var regioni = [];
            regioni.push("Calabria");
            regioni.push("Lazio");
            regioni.push("Lombardia");

            var province = [];
            province.push("Reggio Calabria");
            province.push("Roma");
            province.push("Lecco");

            println("Regioni: " + regioni.toString());
            println("Province: " + province.toString());
}
printMsg();
```

I programmatori JavaScript avranno già intuito il significato di questo codice. In pratica esso definisce la funzione printMsg. Tale funzione crea due array: regioni e province. Dopo aver riempito i due array con i valori opportuni, la funzione ne visualizza il contenuto sulla console. La funzione println, rispetto a print, stampa su una linea e ritorna a capo. Il file test.js termina con la chiamata vera e propria alla funzione prima definita. L'output prodotto è il seguente:

```
Regioni: Calabria,Lazio,Lombardia
Province: Reggio Calabria,Roma,Lecco
```

Chiaramente, il codice JavaScript appena visto non è nulla di eccezionale. Esso, tuttavia, illustra la straordinaria potenzialità offerta da Java 6, ovvero la sua totale integrazione con i linguaggi di script.
Vediamo, ora, come sia possibile caricare una funzione JavaScript e posporre la sua invocazione attraverso codice Java. Tutto ciò può essere fatto grazie all'interfaccia javax.script.Invocable. Nello sviluppo di uno ScriptEngine non è richiesto che esso implementi l'interfaccia Invocable. Tuttavia, se l'engine che vogliamo utilizzare implementa tale interfaccia allora sarà possibile invocare specifici script già "valutati" attraverso il metodo eval. Fortunatamente, il motore di script fornito nativamente da Java 6, ossia Rhino, implementa la suddetta interfaccia. Facciamo subito un esempio. Supponiamo di avere un file JavaScript che contiene tre funzioni. Vogliamo caricare tale file ed invocare una di queste funzioni. Il codice che ci serve è il seguente:

```
ScriptEngineManager engineMgr = new
                                ScriptEngineManager();
ScriptEngine engine =
            engineMgr.getEngineByName("JavaScript");
InputStream is = null;
try
{
  is = new FileInputStream("./scripts/utils.js");
  Reader reader = new InputStreamReader(is);
  engine.eval(reader);
  Invocable invocableEngine = (Invocable) engine;
  invocableEngine.invokeFunction("printLecco");
}
catch (ScriptException ex)
{
  ex.printStackTrace();
}
catch (FileNotFoundException ex)
{
            ex.printStackTrace();
}
catch (NoSuchMethodException ex)
{
            ex.printStackTrace();
}
```

Rispetto al codice dell'esempio precedente non c'è molto da aggiungere. L'unica parte degna di nota è:

```
Invocable invocableEngine = (Invocable) engine;
invocableEngine.invokeFunction("printLecco");
```

La prima riga esegue un cast da ScriptEngine ad Invocable. Questo è possibile perché, come abbiamo detto, l'engine Rhino implementa l'interfaccia Invocable. La seconda riga chiama il metodo invokeFunction. Tale metodo invoca una funzione JavaScript che è stata prima caricata e valutata tramite il metodo eval.
Il file utils.js contiene, invece, codice JavaScript un po' più complicato rispetto al precedente. Questo a riprova della totale integrazione con il linguaggio Java. Vi è, prima di tutto, la definizione JavaScript di una classe:

```
function Comune(regione, provincia, abitanti)
{
            this.regione = regione;
            this.provincia = provincia;
            this.abitanti = abitanti;
}

Comune.prototype.toString = function()
{
```

```
        return "Regione: " + this.regione + "\n" +
                        "Provincia: " +
                        this.provincia + "\n" +
                        "Abitanti: " +
                                this.abitanti;
};
[...]
```

È facile intuire lo scopo della classe Comune. Essa contiene tre campi: regione, provincia ed abitanti. Inoltre, è definito il metodo toString che restituisce una stringa atta a descrivere il contenuto dei membri della classe. La parte restante di utils.js contiene tre funzioni così definite:

```
function printReggio()
{
        var reggioCalabria = new
    Comune("Calabria", "Reggio Calabria", "180.353");
        print("Info su Reggio Calabria: " + "\n" +
                        reggioCalabria.toString());
}

function printRoma()
{
        var roma = new Comune("Lazio", "Roma",
                                "3.700.424");
        print("Info su Roma: " + "\n" +
                                roma.toString());
}

function printLecco()
{
        var lecco = new Comune("Lombardia",
                        "Lecco", "311.452");
    print("Info su Lecco: " + "\n" + lecco.toString());
}
```

L'output prodotto dall'invocazione di print Lecco, attraverso invokeFunction, è il seguente:

```
Info su Lecco:
Regione: Lombardia
Provincia: Lecco
Abitanti: 311.452
```

Esso non è altro che la rappresentazione in stringa dell'oggetto Lecco definito in printLecco.
Come abbiamo visto, quindi, grazie alle nuove potenzialità di Mustang è possibile:

- Eseguire script rappresentati tramite una stringa (attraverso il metodo eval di ScriptEngine).
- Compilare ed eseguire codice JavaScript residente in un file esterno (attraverso un overload del metodo eval di ScriptEngine).
- Compilare ed, in un secondo momento, invocare funzioni o metodi JavaScript (attraverso il metodo eval di ScriptEngine e l'interfaccia Invocable).

JAVA DA CODICE JAVASCRIPT

Finora abbiamo visto come è possibile invocare funzioni e metodi JavaScript da codice Java. Questo paragrafo, tuttavia, affronterà il problema opposto, ossia il passaggio di oggetti Java a funzioni JavaScript. La via più corta ed intuitiva per raggiungere questo risultato è utilizzare il metodo put dell'interfaccia ScriptEngine. Tale metodo ha la seguente signature:

```
public void put(String key, Object value)
```

L'interfaccia ScriptEngine, a sua volta, usa javax.script.Binding per mantenere le coppie chiave-valore ed effettuare il binding vero e proprio.
Facciamo subito un esempio in cui degli oggetti Java vengono usati all'interno di uno script. Guardate il seguente codice:

```
List<Person> people = new ArrayList<Person>();
people.add(new Person("Alessandro", "Lacava",
                                "Lecco", 31));
people.add(new Person("Domenico", "Mordà",
                                "Lecco", 28));
people.add(new Person("Antonino", "Catanese",
                                "Lecco", 29));

ScriptEngineManager engineMgr = new
                        ScriptEngineManager();
ScriptEngine engine =
        engineMgr.getEngineByName("JavaScript");
engine.put("peopleVar", people);
InputStream is = null;
try
```

JSR

Le Java Specification Request (JSR) sono documenti formali che descrivono proposte su specifiche e tecnologie da aggiungere alla piattaforma Java. Prima che una JSR diventi "final" viene sottoposta a varie revisioni ed infine ai voti del comitato esecutivo del JCP (Java Community Process). Quando una JSR diventa finale, viene fornita un'implementazione di riferimento della specifica. Lo stesso JCP è definito da una JSR, ossia JSR 215. Attualmente vi sono oltre 300 JSR.

```
{
        is = new
    FileInputStream("./scripts/javaToJavaScript.js");
        Reader reader = new
                InputStreamReader(is);
        engine.eval(reader);
}
catch (ScriptException ex)
{
ex.printStackTrace();
}
catch (FileNotFoundException ex)
{
ex.printStackTrace();
}
```

Come potete vedere, viene creata una lista di oggetti di tipo Person. Tale classe è un semplice bean con i suoi getter e setter ed una ridefinizione del metodo toString che riporto di seguito:

```
public String toString()
{
        return "Nome: " + firstName +
System.getProperty("line.separator") +
                        "Cognome: "
                        + lastName +
System.getProperty("line.separator") +
                        "Città: " +
                              city +
System.getProperty("line.separator") +
                        "Età: " +
                              age;
}
```

Il significato del resto del codice lo abbiamo già spiegato precedentemente. Vi è una sola novità, ossia la chiamata al metodo put:

```
engine.put("peopleVar", people);
```

Tale codice non fa altro che "bindare" l'oggetto Java people, ossia la lista, alla variabile peopleVar che sarà utilizzata da codice JavaScript. A tal proposito vediamo il contenuto del file javaToJavaScript.js:

```
// Usa peopleVar che è stata "bindata"
// nel codice Java
var persone = peopleVar.toArray();
function displayPeople()
{
        for(i in persone)
        {
                var persona = persone[i];
                println("Persona: \n" +
                        persona.toString());
        }
}
displayPeople();
```

La prima riga converte l'oggetto peopleVar in un array e lo referenzia tramite la variabile persone. In seguito il codice definisce una funzione, displayPeople, e la richiama. La funzione non fa altro che ciclare attraverso l'array e richiamare il metodo toString definito nella classe Person. L'output prodotto è il seguente:

```
Persona:
Nome: Alessandro
Cognome: Lacava
Città: Lecco
Età: 31
Persona:
Nome: Domenico
Cognome: Mordà
Città: Lecco
Età: 28
Persona:
Nome: Antonino
Cognome: Catanese
Città: Lecco
Età: 29
```

Come potete vedere, vi è una totale integrazione tra linguaggio di script e Java. Infatti, da codice JavaScript invochiamo, in modo del tutto trasparente, un metodo definito in una classe Java, ossia toString della classe Person. Inoltre, come abbiamo visto dalla prima riga di codice dello script, il binding tra la lista e la variabile peopleVar è avvenuto senza problemi. Da questo semplice esempio traspare tutta la potenza delle JSR 223, ossia il supporto di Java per i linguaggi di scripting.
È rimasto un altro problemino da risolvere. Vogliamo, ora, passare oggetti Java come parametri di una funzione JavaScript. In altre parole, nell'esempio precedente abbiamo richiamato una funzione JavaScript, displayPeople, che non prendeva alcun parametro. Quello che vogliamo fare ora, è poter richiamare una funzione JavaScript che accetta parametri e tali parametri devono essere valorizzati con oggetti Java. Per risolvere questo tipo di problema ci viene incontro un'interfaccia già vista, ossia Invocable. Attraverso il metodo invokeFunction di questa interfaccia, è possibile invocare una funzione JavaScript e passare un numero arbitrario di parametri. La signature del metodo in questione è la seguente:

```
public Object invokeFunction(String name,
                                Object... args)
```

Vediamo, ora, un esempio che utilizza questo metodo per invocare una funzione JavaScript che riceve in ingresso dei parametri. A tal proposito riprendiamo l'esempio precedente apportando le opportune modifiche. Ecco il codice:

```
List<Person> people = new ArrayList<Person>();
people.add(new Person("Alessandro", "Lacava",
                                    "Lecco", 31));
people.add(new Person("Domenico", "Mordà",
                                    "Lecco", 28));
people.add(new Person("Antonino", "Catanese",
                                    "Lecco", 29));

ScriptEngineManager engineMgr = new
                              ScriptEngineManager();
ScriptEngine engine =
              engineMgr.getEngineByName("JavaScript");

InputStream is = null;
try
{
        is = new
  FileInputStream("./scripts/javaToJSWithParams.js");
        Reader reader = new
                              InputStreamReader(is);
        engine.eval(reader);
    // LE DUE RIGHE CHE SEGUONO SONO LE
    // PIU' IMPORTANTI
        Invocable invocableEngine = (Invocable)
                                            engine;
invocableEngine.invokeFunction("displayPeople",
                                            people);
}
catch (ScriptException ex)
{
  ex.printStackTrace();
}
catch (FileNotFoundException ex)
{
        ex.printStackTrace();
}
catch (NoSuchMethodException ex)
{
        ex.printStackTrace();
}
```

Come potete vedere, non è cambiato molto rispetto all'esempio precedente. In questo caso, invece di utilizzare il metodo put di ScriptEngine usiamo l'interfaccia Invocable. In particolare, tramite il suo metodo invokeFunction, invochiamo la funzione JavaScript displayPeople passandogli la lista people come parametro. Il metodo invokeFunction può sollevare un'eccezione del tipo NoSuchMethod Exception. Per tale motivo abbiamo aggiunto un blocco catch atto ad "acchiappare" questa eventuale eccezione. Rimane da vedere il contenuto del file javaToJSWithParams.js. Lo riporto di seguito:

```
function displayPeople(people)
{
        var persone = people.toArray();
        for(i in persone)
        {
                var persona = persone[i];
                println("Persona: \n" +
                              persona.toString());
        }
}
```

Potete notare la forte somiglianza che tale funzione ha con quella definita nell'esempio precedente. In questo caso, tuttavia, people è definito come parametro della funzione e non come variabile esterna. Inoltre, non vi è più la necessità di invocare la funzione displayPeople in quanto viene invocata da codice Java. Questo ci permette di scrivere codice più elegante, poiché possiamo definire le funzioni JavaScript in un file esterno e poi invocarle, all'occorrenza, da codice Java. L'output di questo esempio, che non riporto per brevità, è analogo a quello precedente. Riassumendo, per passare oggetti Java in un contesto di script vi sono, principalmente, due strade:

- Utilizzare il metodo put dell'interfaccia ScriptEngine.
- Effettuare un cast da ScriptEngine ad Invocable e chiamare il metodo invokeFunction sull'oggetto "castato". Questa seconda strada è utilizzata nel caso in cui bisogna passare dei parametri alla funzione JavaScript.

CONCLUSIONI

In quest'articolo abbiamo visto una delle nuove feature di Java Platform, Standard Edition versione 6. Abbiamo analizzato le principali peculiarità di JSR 223, ossia il supporto di Java per i linguaggi di scripting. In particolare, si è illustrato come integrare codice JavaScript in Java e viceversa. Questo offre nuove potenzialità per lo sviluppo di software molto flessibile e potente. Ad esempio è possibile sviluppare, in JavaScript, le funzionalità necessarie per effettuare chiamate Ajax ed invocare tali funzioni direttamente da Java o addirittura incapsulare tali API creando una libreria in modo da rendere il tutto trasparente allo sviluppatore Java che utilizzerà le nostre API. Le possibilità sono innumerevoli quindi non ci resta che far lavorare la nostra fantasia.

Alessandro Lacava

VISUAL STUDIO LO SVILUPPO IO

CERTO, I WIZARD SONO UNA GRAN COMODITÀ. SPESSO PERÒ LE CLASSI PRODOTTE SONO DIFFICILMENTE GESTIBILI. POTREMMO SCRIVERE DI VOLTA IN VOLTA IL CODICE A MANO. OPPURE PROGETTARE UN NOSTRO GENERATORE DI CODICE. ECCO COME...

Credo che molti programmatori siano d'accordo con me nell'affermare che l'80% della nostra attività consiste in operazioni ripetitive e che molte di queste operazioni riguardino il codice necessario alla connessione con i database. Paradossalmente l'avanzare del paradigma della programmazione orientata agli Oggetti ha complicato invece di semplificare le cose: mettiamola come vi pare ma una Classe resta sempre una cosa ben diversa da una Tabella di database, il codice SQL è sempre e comunque un "alieno" nei linguaggi di programmazione Object Oriented, difficile da mantenere e aggiornare di pari passo con lo sviluppo del programma. Non è quindi un caso che molti sforzi dei produttori di IDE e tools vari siano indirizzati a rendere trasparenti le operazioni di *Mapping* del database in Classi. Anche Microsoft ha seguito questa linea con i Dataset Tipizzati presenti in Visual Studio .NET fin dalle prime edizioni.

I DATASET TIPIZZATI DI VISUAL STUDIO

I Dataset Tipizzati di Visual Studio sono un promettente strumento che però, a mio avviso, delude in gran parte le aspettative, vediamo perché. Proviamo a costruire un semplicissimo Dataset Tipizzato prendendo come riferimento il database di esempio di SQL Server chiamato Northwind. In un nuovo progetto di tipo "Console Application" di Visual Studio aggiungiamo un nuovo elemento *DataSet*. Aggiungiamo la connessione al nostro database nella finestra "Esplora Server" e trasciniamo una tabella (ad esempio *Customers*) nella finestra di progettazione principale del componente DataSet. Nel progetto sembra che non sia cambiato niente, in realtà se nella finestra "Esplora soluzioni" attiviamo l'opzione "Mostra tutti i file" vediamo (**fig. 1**) che, sotto il file del Dataset, sono comparsi altri file tra cui un file di codice generato automaticamente.

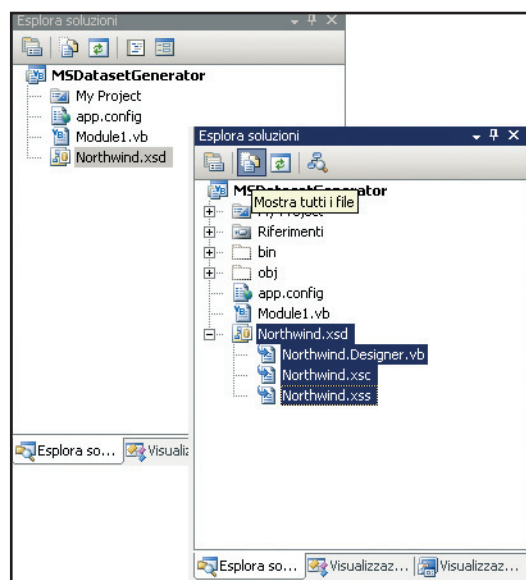



Figura 1: finestra di progettazione principale del componente

Se andiamo a verificare con il Diagramma Classi il codice creato scopriamo che è stato creata una nuova classe **Northwind** derivata da Dataset che, all'interno, ha le seguenti classi nidificate:

- **CustomersDataTable** – derivata da Data Table che rappresenta la tabella Customers
- **CustomersRow** – derivata da DataRow che rappresenta la riga della tabella Customers con i membri (corrispondenti ai campi) esplicitamente dichiarati.
- **CustomersRowChangeEvent** e **Customers RowChangeEventHandler** – che rappresentano l'evento RowChange per la CustomersDataTable.

Oltre a **Northwind**, sotto allo spazio di nomi *NorthwindTableAdapters*, è stata poi creata la classe *CustomersTableAdapter* che rappresenta un DataAdapter specificamente rivolto a **CustomersDataTable**.
L'utilizzo di queste classi, in sé non è difficile, poniamo di voler stampare a video nomi e indi-

rizzi dei clienti, una possibile implementazione è la seguente :

```
Sub Main()
    ' Creo la connessione
    Dim conn As New SqlConnection("Data
        Source=Localhost;Initial
        Catalog=Northwind;Integrated Security=True")
    ' Creo l'istanza dell'Adapter
    Dim adp As New
        NorthwindTableAdapters.CustomersTableAdapter
    adp.Connection = conn
    ' Creo una nuova tabella
    Dim cust As Northwind.CustomersDataTable
        = adp.GetData()
    ' Richiamo il metodo Fill
    adp.Fill(cust)
    For Each r As Northwind.CustomersRow
        In cust.Rows
        ' Stampa a video i risultati
        Console.WriteLine("{0} - {1}",
            r.CompanyName, r.Address)
    Next
End Sub
```

Fin qui tutto bene, ma se voglio includere un campo che può avere valori nulli, come il campo *Region* (che Visual Studio ha mappato come la proprietà *_Region* della classe Northwind. CustomersRow) e scrivo quindi:

```
Console.WriteLine("{0} - {1} {2}",
    r.CompanyName, r.Address, r._Region)
```

Eseguendo di nuovo il programma si riscontra subito un'Eccezione. Se andiamo a vedere nel codice generato da Visual Studio scopriamo il motivo dell'errore nella proprietà _Region :

```
Public Property _Region() As String
    Get

        Try
            Return
                CType(Me(Me.tableCustomers.RegionColumn),
                String)
        Catch e As
            System.InvalidCastException
            Throw New
                System.Data.StrongTypingException("Il valore
                della colonna 'Region' nella tabella 'Customers' è
                DBNull.", e)
        End Try
    End Get
    Set
Me(Me.tableCustomers.RegionColumn) = value
    End Set
End Property
```

In pratica succede che nel Get della proprietà non viene controllato se il valore della colonna è nullo, ma viene generata un'eccezione se non riesce a convertirlo in stringa.
Correggere il problema sarebbe facilissimo, basterebbe scrivere del codice di controllo tipo :

```
Public Property _Region() As String
    Get
        Try
            If Not Me.Is_RegionNull Then
                Return
                    CType(Me(Me.tableCustomers.RegionColumn),
                    String)
            Else
                Return ""
            End If
        Catch e As
            System.InvalidCastException
            Throw New
                System.Data.StrongTypingException("Il valore
                della colonna 'Region' nella tabella 'Customers' è
                DBNull.", e)
        End Try
    End Get
    Set
Me(Me.tableCustomers.RegionColumn) = value
    End Set
End Property
```

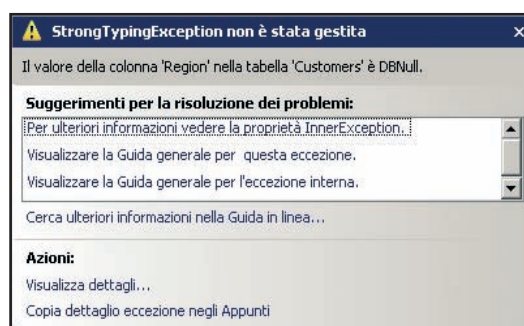



***Figura 2** Exception su valori nulli del campo*

tuttavia questa operazione non è affatto sicura poiché questo codice è generato automaticamente e qualunque modifica viene sovrascritta se modifichiamo, anche minimamente, il DataSet originario.
Quindi, per evitare eccezioni, siamo costretti a scrivere il codice di controllo, per i campi che possono contenere valori nulli, direttamente in fase di utilizzo del DataSet Tipizzato, in questo modo:

```
Sub Main()
    Dim conn As New SqlConnection("Data
        Source=SMELZO;Initial
        Catalog=Northwind;Integrated Security=True")
```

```
    Dim cmd As SqlCommand =
                          conn.CreateCommand
    Dim adp As New
    NorthwindTableAdapters.CustomersTableAdapter
    adp.Connection = conn
    Dim cust As Northwind.CustomersDataTable
                                = adp.GetData()
    adp.Fill(cust)
    For Each r As Northwind.CustomersRow
                                   In cust.Rows
        Dim CompanyName As String =
                                  r.CompanyName
        Dim Address As String = ""
        If Not r.IsAddressNull Then
            Address = r.Address
        End If
        Dim Region As String = ""
        If Not r.Is_RegionNull Then
            Region = r._Region
        End If
        Console.WriteLine("{0} - {1} {2}",
                   CompanyName, Address, Region)
    Next
    Console.ReadLine()
End Sub
```

Beh, immaginatevi di farlo per tabelle con una decina di colonna e scoprirete che, alla fine, le cose sono più semplici utilizzando direttamente le classi ADO come ad esempio:

```
Private Function ReadRow(ByVal r As DataRow,
                ByVal fieldName As String) As Object
    If Not r.IsNull(fieldName) Then
        Return r(fieldName)
    End If
    Return Nothing
End Function
Sub Main()
    Dim conn As New SqlConnection("Data
                         Source=SMELZO;Initial
     Catalog=Northwind;Integrated Security=True")
    Dim adp As New SqlDataAdapter("Select *
                          From Customers", conn)
    Dim tb As New DataTable("Customers")
    adp.Fill(tb)
    For Each r As DataRow In tb.Rows
        Console.WriteLine("{0} - {1} {2}",
          ReadRow(r, "CompanyName"), ReadRow(r,
              "Address"), ReadRow(r, "Region"))
    Next
    Console.ReadLine()
End Sub
```

Il problema dei Dataset Tipizzati si può riassumere quindi in: un'infrastruttura pesante da utilizzare e con minimi vantaggi pratici rispetto all'accesso diretto al database.

Certo ci sono anche delle alternative come NHibernate, il porting su .NET del progetto Hibernate in Java, ma anche lì in termini di pesantezza e complessità non scherziamo affatto!

Tornando invece alla soluzione proposta da Microsoft, vediamo che, anche se l'implementazione può lasciare a desiderare, l'idea di base è molto valida : partire da una schema per arrivare alla generazione automatica del codice.

Se ci riflettiamo un attimo quello che ci servirebbe è costruirci un nostro, personalissimo, generatore di codice che lo crei come vogliamo noi.

GENERATORI DI CODICE IN VISUAL STUDIO

Il meccanismo della generazione del codice in Visual Studio non è proprio soltanto dei Dataset ma è presente anche in molti altri casi : per i file di risorse, per la configurazione, per il collegamento ai Webservice ecc…, i componenti che svolgono questo lavoro prendono il nome di Custom Tools.

Un Custom Tool non è altro che un componente COM che implementa l'interfaccia IVSingleFileGenerator (Visual Studio infatti è largamente basato su tecnologia COM), questa e le altre interfacce necessarie ad interagire con Visual Studio possono essere implementate da codice .NET esposto a COM attraverso l'interoperabilità, tuttavia questo lavoro (non semplicissimo) ci viene risparmiato scaricando la libreria di esempio **BaseCode GeneratorWithSite** sul sito www.gotdotnet. com .

La libreria, della stessa Microsoft, è scritta in C# per i Framework 1.0 e 1.1 tuttavia possiamo tranquillamente convertirla e compilarla con Visual Studio 2005.

In pratica la sua funzione è quella di implementare IVSingleFileGenerator ed altre interfacce utili allo sviluppo del Custom Tool.

Ma prima di vedere come utilizzare la libreria, vediamo cos'è un Custom Tool.

Se riprendiamo l'esempio iniziale con il DataSet Tipizzato e, nel progetto, in Esplora soluzioni, andiamo a selezionare il file del dataset (nel nostro caso Northwind.xsd) noteremo che nella Finestra Proprietà appare anche la voce Strumento Personalizzato (se Visual Studio è in inglese ovviamente sarà Custom Tool) e, in questo caso, è valorizzato a MSDataSetGenerator, che è appunto il generatore di codice associato ai Dataset. Il nome indicato come valore della proprietà corrisponde in realtà a una classe COM registrata nel sistema.

Infatti se andiamo ad esplorare con Regedit il

registro di configurazione del sistema vediamo tutti i Custom Tools di Visual Studio elencati sotto la chiave HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\VisualStudio\8.0\Generators

Qui troviamo tre sottochiavi che hanno come nome delle GUID e rappresentano i diversi linguaggi installati:

- {164b10b9-b200-11d0-8c61-00a0c91e29d5} – Visual Basic
- {E6FDF8B0-F3D1-11D4-8576-0002A516ECE8} – J#
- {fae04ec1-301f-11d3-bf4b-00c04f79efbc} – C#

Sotto ognuna di queste chiavi troviamo i generatori specifici per quel linguaggio, se andiamo ad esempio sotto la chiave MSDataSetGenerator troviamo tre valori:

- Default – Descrizione del generatore
- CLSID – Che indica appunto il CLSID con cui è registrata la classe COM che fornisce il servizio
- GeneratesDesignTimeSource – un valore DWORD valorizzato a 1 che indica che il servizio è attivo

Da questo si desume che, per creare un altro Custom Tool occorre:

- Creare una classe COM che implementa le necessarie interfacce, almeno una per ogni linguaggio che intendiamo gestire.
- Creare le chiavi e i valori simili a quelli che abbiamo visto nel registro di configurazione.
- Indicare il nome che abbiamo scelto in Visual Studio nella proprietà *Strumento Personalizzato* di un file che ne contiene le definizioni.

Ad esempio, se avessimo creato un Generatore di codice che chiameremo MioGeneratore e che genera sia codice VB che C# :

Creiamo le chiavi HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Generators\{164b10b9-b200-11d0-8c61-00a0c91e29d5}\ MioGeneratore per Visual Basic
e HKEY_LOCAL_MACHINE\SOFTWARE \Microsoft\VisualStudio\8.0\Generators\{fae04ec1-301f-11d3-bf4b-00c04f79efbc}\ MioGeneratore per C#

Sotto ognuna di queste chiavi mettiamo:

- Una descrizione come valore predefinito
- La GUID per la classe che gestisce lo specifico linguaggio sotto CLSID
- Il valore 1 DWORD chiamato Generates DesignTimeSource

Mi rendo conto però che vista così sembra una cosa complessa per cui è meglio passare subito a un esempio concreto.

IL NOSTRO PRIMO GENERATORE

Vediamo quindi come fare a creare un nostro primo, semplice, generatore di codice.
Per prima cosa scarichiamo e compiliamo in VS2005 la libreria **BaseCodeGeneratorWithSite**.
Nella compilazione bisogna tener presente che l'oggetto deve essere esposto a COM per cui, nelle proprietà del progetto è necessario sia fornire l'assembly di un nome sicuro che registrarlo per la compatibilità COM.
Poi creiamo un nuovo progetto libreria in Visual Basic o C#, anch'esso esposto a COM come mostrato in figura 8.
Lo scheletro della classe che gestisce il servizio si presenterà pressappoco così in VB:

```
Imports System.Runtime.InteropServices
Imports CustomToolGenerator

<Guid("9695a0c9-7a6a-49d6-b697-
                    590cd6177bba")> _
Public Class MioGeneratore
    Inherits BaseCodeGeneratorWithSite

    Protected Overrides Function
        GenerateCode(ByVal inputFileName As String,
            ByVal inputFileContent As String) As Byte()

    End Function
End Class
```

o così in C#:

```
using System.Runtime.InteropServices;
using CustomToolGenerator;
[Guid("9695a0c9-7a6a-49d6-b697-
                    590cd6177bba")]
public class MioGeneratore :
                    BaseCodeGeneratorWithSite
{
        protected override byte[]
          GenerateCode(string inputFileName, string
                    inputFileContent)
        {
        }
}
```

LA LIBRERIA BASECODEGENERATORWITHSITE

La libreria BaseCodeGeneratorWithSite, di cui parliamo nell'articolo è liberamente scaricabile all'indirizzo http://www.gotdotnet.com/Community/UserSamples/Details.aspx?SampleGuid=4AA14341-24D5-45AB-AB18-B72351D0371C

Com’è evidente tutta l’operazione si concentra nella funzione GenerateCode che fornisce due parametri:

- inputFileName – il nome del file da cui viene generato il codice (uno schema XSD, un file XML o di testo o qualunque altra cosa ci venga in mente di usare)
- inputFileContent – il contenuto in forma di stringa di questo file

Il risultato della funzione è la matrice di byte che verrà scritta nel file generato.
Se, ad esempio, abbiamo un file di input chiamato *schema.txt* il file generato sarà schema.vb o *schema.cs* a seconda che si applichi il generatore in un progetto VB o C#.
Ultima cosa da notare è che la classe è dotata dell’attributo GUID, questo è fondamentale per essere vista da COM (e quindi da Visual Studio) e per essere registrata nel Registry di Windows.
Ogni classe Generatore della nostra libreria gestirà un linguaggio differente, per cui bisognerà approntare una classe per ogni linguaggio per cui :

```
<Guid("9695a0c9-7a6a-49d6-b697-
                        590cd6177bba")> _
Public Class MioGeneratoreVB
    Inherits BaseCodeGeneratorWithSite

    Protected Overrides Function GenerateCode(ByVal
    inputFileName As String, ByVal inputFileContent As
                                String) As Byte()
    End Function
End Class
<Guid("77d5fe6d-b886-474f-9ca5-af2bb32429ff")> _
Public Class MioGeneratoreCS
    Inherits BaseCodeGeneratorWithSite

    Protected Overrides Function GenerateCode(ByVal
    inputFileName As String, ByVal inputFileContent As
                                String) As Byte()

    End Function
End Class
```

con la prima classe gestiremo la generazione di codice VB, con la seconda la generazione di codice C#.
Una cosa che dobbiamo decidere è poi la forma che deve avere il file di Input, quello cioè che viene letto per produrre il codice.
In questa prima fase non vogliamo ancora complicare troppo le cose, quindi rimandiamo per il momento la generazione di codice per il database, e ci limitiamo ad un obbiettivo più limitato ma un po’ più utile del classico “Hallo World”.

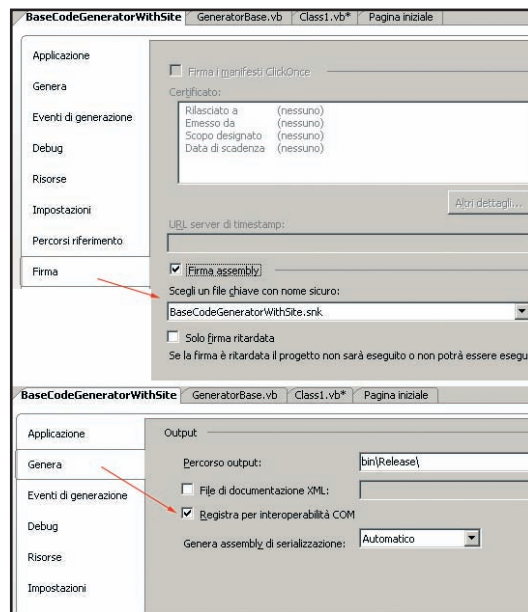



Figura 8: Proprietà del progetto per l’esposizione a COM

Ci riproporremmo quindi di partire da un semplice file XML che definisce lo scheletro di una classe il quale verrà letto e trasformato in codice. Il file, che chiameremo *prova.xml*, sarà pressappoco così:

```
<?xml version="1.0" encoding="utf-8" ?>
<class name="Customer" namespace="Northwind">
  <property name="CustomerID"
                  type="Integer"></property>
  <property name="CompanyName"
                  type="String"></property>
  <property name="ContactName"
                  type="String"></property>
  <property name="ContactTitle"
                  type="String"></property>
  <property name="Address"
                  type="String"></property>
  <property name="City" type="String"></property>
  <property name="Region"
                  type="String"></property>
  <property name="PostalCode"
                  type="String"></property>
  <property name="Country"
                  type="String"></property>
  <property name="Phone"
                  type="String"></property>
  <property name="Fax" type="String"></property>
</class>
```

Definiamo quindi, nel codice, due classi che possano servire da contenitore dei dati deserializzati:

```
Imports System.Xml
Imports System.Xml.Serialization
```

```
Imports System.Collections.Generic
<XmlRoot("class")> _
Public Class ClassModel
    <XmlAttribute("name")> Public Name As String
    <XmlAttribute("namespace")> Public
                                ClassNamespace As String

    Private _Properties As List(Of PropertyModel)
    <XmlElement("property")> _
    Public Property Properties() As List(Of
                                        PropertyModel)
        Get
            If _Properties Is Nothing Then
                _Properties = New List(Of PropertyModel)
            End If
            Return _Properties
        End Get
        Set(ByVal Value As List(Of PropertyModel))
            _Properties = Value
        End Set
    End Property
End Class
Public Class PropertyModel
    <XmlAttribute("name")> Public Name As String
    <XmlAttribute("type")> Public TypeName As
                                            String
End Class
```

Definiamo ora una classe astratta che rappresenta il ciclo di trasformazione, dall'acquisizione dell'input fino alla produzione dell'output, la chiameremo CodeWriterBase :

```
Imports System.Xml
Imports System.Xml.Serialization
Public MustInherit Class CodeWriterBase
    Protected Sb As System.Text.StringBuilder
    Protected Model As ClassModel
    Protected Function GetInput(ByVal s As String) As
                                            ClassModel
        Dim serializer As New
                    XmlSerializer(GetType(ClassModel))
        Dim textReader As New IO.StringReader(s)
        Return serializer.Deserialize(textReader)
    End Function
    Protected Sub Init(ByVal input As String)
        Sb = New System.Text.StringBuilder
        Model = GetInput(input)
    End Sub
    Public MustOverride Function Generate(ByVal input
                                  As String) As Byte()
End Class
```

La classe servirà da base per le implementazioni nei vari linguaggi e presenta un metodo per deserializzare l'input e per inizializzare le variabili interne.
Si tratta, a questo punto, di implementare la classe CodeWriter per Visual Basic e quella per C#

COME GENERARE IL CODICE

La questione da risolvere, a questo punto, è : quale strategia seguire per generare il codice?

Template

Come abbiamo visto, in realtà si tratta, alla fine, di produrre una matrice di byte che è poi la stringa di codice convertita in byte, una cosa pressappoco come:

```
Dim StrCode As String = "Public Class Prova" &
                                        vbNewline & _
StrCode &= "End Class"
Return
        System.Text.Encoding.ASCII.GetBytes(StrCode)
```

ovvero inserire tutto il codice in una stringa e convertirlo in byte con la codifica appropriata.
Naturalmente a nessuno verrebbe in mente di generare centinaia di righe di codice concatenando una stringa.
La tecnica prevede quindi la creazione di alcuni template come, ad esempio, nel caso precedente:

```
Public Class {ClassName}
End Class
```

che, in fase di generazione, vengono letti sostituendo ai segnaposti i valori appropriati, come in :

```
Dim StrCode As String =
    ReadTemplate("Class.txt").Replace("{ClassName}"
                                          , "Prova")
Return
        System.Text.Encoding.ASCII.GetBytes(StrCode)
```

Questa soluzione è senz'altro molto lineare e presenta vantaggi indubbi di chiarezza di editazione e di mantenibilità dei template, l'unico svantaggio è che, dovendo generare codice per più linguaggi, siamo costretti ad avere più template paralleli, uno per ogni linguaggio, che devono essere costantemente allineati se non ci vogliamo ritrovare con versioni differenti.

CODEDOM

.NET però mette a disposizione anche una soluzione alternativa rappresentata dal CodeDom, il CodeDom rappresenta in maniera astratta tutti i costrutti possibili utilizzabili in un programma.

Per rappresentare, ad esempio, la dichiarazione di un Namespace con una Classe che contiene un Campo *_ID* e una proprietà *ID* di tipo Intero, utilizzando il CodeDom dovremmo esprimerla come:

```
Imports System.CodeDom
Public Class CodeDomTest
    Private Shared Function
            CreateCodeNamespace() As CodeNamespace
        Dim ns As New CodeNamespace("Prova")
        Dim cls As New CodeTypeDeclaration("Test")
        cls.IsClass = True
        Dim f As New
          CodeMemberField(GetType(Integer).FullName,
                                              "_ID")
        cls.Members.Add(f)
        Dim m As New CodeMemberProperty
        m.Name = "ID"
        m.HasGet = True
        m.HasSet = True
        m.Type = New
        CodeTypeReference(GetType(Integer).FullName)
        Dim privateFieldRef As New
                        CodeFieldReferenceExpression(New
                    CodeThisReferenceExpression(), "_ID")
        Dim ReturnExpr As New
          CodeMethodReturnStatement(privateFieldRef)
        m.GetStatements.Add(ReturnExpr)

        Dim rightExpr As New
              CodePropertySetValueReferenceExpression
        Dim setExpression As New
        CodeAssignStatement(privateFieldRef, rightExpr)
        m.SetStatements.Add(setExpression)
        cls.Members.Add(m)
        ns.Types.Add(cls)
        Return ns
    End Function
End Class
```

La funzione *CreateCodeNamespace* crea cioè un oggetto *CodeNamespace* definendo al suo interno la classe, i campi, le proprietà, le espressioni ecc…
Come avete visto, la complessità del codice non è di poco conto, il vantaggio è che poi sarà possibile interpretare l'oggetto con qualsiasi generatore di codice di .NET, come in :

```
    Shared Sub GenerateCodeNamespace()
        Console.WriteLine("VB CODE:")
        Dim VbProvider As New
                  Microsoft.VisualBasic.VBCodeProvider
        Dim options As New
                        Compiler.CodeGeneratorOptions
VbProvider.GenerateCodeFromNamespace(CreateCode
                      Namespace, Console.Out, options)
        Console.WriteLine()
        Console.WriteLine("C# CODE:")
        Dim CsProvider As New
                  Microsoft.CSharp.CSharpCodeProvider

CsProvider.GenerateCodeFromNamespace(CreateCode
                      Namespace, Console.Out, options)
    End Sub
```

dove, appunto, lo stesso oggetto viene prima interpretato con il provider per VB.NET e poi con quello per C# in modo da produrre un output tipo:

```
VB CODE:
Namespace Prova
    Public Class Test
        Private _ID As Integer
        Private Property ID() As Integer
            Get
                Return Me._ID
            End Get
            Set
                Me._ID = value
            End Set
        End Property
    End Class
End Namespace

C# CODE:
namespace Prova {
    public class Test {
        private int _ID;
        private int ID {
            get {
                return this._ID;
            }
            set {
                this._ID = value;
            }
        }
    }
```

Il vantaggio è evidentemente quello di avere un sorgente univoco che poi può essere riprodotto in qualsiasi linguaggio, di contro scrivere codice complesso utilizzando il CodeDom è una vera sofferenza! Quindi, almeno nel nostro caso, utilizzeremo la tecnica dei template, anzi, poiché il codice che è necessario generare è veramente minimo, ricorreremo semplicemente a uno StringBuilder.

LE CLASSI CODEWRITER

L'implementazione delle classi CodeWriter sarà quindi:

```
Public Class VBCodeWriter
    Inherits CodeWriterBase

    Public Overrides Function Generate(ByVal input As String) As Byte()
        Init(input)
        Sb.AppendFormat("Public Class {0}", Model.Name).AppendLine()
        For Each prop As PropertyModel In Model.Properties
            Sb.AppendFormat("Private _{0} As {1}", prop.Name, prop.TypeName).AppendLine()
            Sb.AppendFormat("Public Property {0} () As {1}", prop.Name, prop.TypeName).AppendLine()
            Sb.AppendLine("Get")
            Sb.AppendFormat("Return _{0}", prop.Name).AppendLine()
            Sb.AppendLine("End Get")
            Sb.AppendFormat("Set (ByVal Value As {0})", prop.TypeName).AppendLine()
            Sb.AppendFormat("_{0}=Value", prop.Name).AppendLine()
            Sb.AppendLine("End Set")
            Sb.AppendLine("End Property")
        Next
        Sb.AppendLine("End Class")
        Dim s As String = Sb.ToString
        Return GetBytes(s)
    End Function
End Class

Public Class CSCodeWriter
    Inherits CodeWriterBase
    Public Overrides Function Generate(ByVal input As String) As Byte()
        Init(input)
        Sb.AppendFormat("public class {0}", Model.Name)
        Sb.AppendLine("{")
        For Each prop As PropertyModel In Model.Properties
            Sb.AppendFormat("private {0} _{1};", prop.TypeName, prop.Name).AppendLine()
            Sb.AppendFormat("public  {0} {1}", prop.TypeName, prop.Name)
            Sb.AppendLine("{")
            Sb.AppendLine("get {")
            Sb.AppendFormat("return  _{0};", prop.Name).AppendLine()
            Sb.AppendLine("}")
            Sb.AppendLine("set {")
            Sb.AppendFormat("_{0}=value;", prop.Name).AppendLine()
            Sb.AppendLine("}")
            Sb.AppendLine("}")
        Next
        Sb.AppendLine("}")
        Dim s As String = Sb.ToString
        Return GetBytes(s)
    End Function
End Class
```

A questo punto abbiamo un CodeWriter per VB e uno per C# che potremo utilizzare all'interno dei generatori:

```
<Guid("9695a0c9-7a6a-49d6-b697-590cd6177bba")> _
Public Class ClassGeneratorVB
    Inherits BaseCodeGeneratorWithSite
    Protected Overrides Function GenerateCode(ByVal inputFileName As String, ByVal inputFileContent As String) As Byte()
        Dim writer As New ClassGenerator.VBCodeWriter()
        Return writer.Generate(inputFileContent)
    End Function
End Class
<Guid("77d5fe6d-b886-474f-9ca5-af2bb32429ff")> _
Public Class ClassGeneratorCS
    Inherits BaseCodeGeneratorWithSite
    Protected Overrides Function GenerateCode(ByVal inputFileName As String, ByVal inputFileContent As String) As Byte()
        Dim writer As New ClassGenerator.CSCodeWriter()
        Return writer.Generate(inputFileContent)
    End Function
End Class
```

Compiliamo quindi il nostro di Class Generator e registriamo le interfacce nel Registry.
Per testare il funzionamento apriamo un nuovo progetto VB o C# e inseriamo un nuovo file XML come quello di esempio che abbiamo visto in precedenza. Poi sulle proprietà del file, alla voce *Strumento Personalizzato* indichiamo il nome "ClassGenerator" e, se tutto è andato a buon fine, se attiviamo la visualizzazione di tutti i file, noteremo che sotto l'icona del file XML compare ora il file di codice generato con la classe disponibile per essere utilizzata.

CONCLUSIONI

Ci fermiamo qui. Abbiamo visto come costruire un Generatore di Codice personalizzato e, anche se ancora non è un generatore per il mapping di un database, è comunque la base tecnica per andare avanti. Nel prossimo numero tratteremo , nello specifico, le problematiche della scrittura di un Custom Tool per il DB Mapping.

Francesco Smelzo

L'AUTORE

Francesco Smelzo è specializzato nello sviluppo in ambiente Windows con particolare riferimento ad applicazioni in ambiente .NET sia web-oriented che desktop. Il suo sito web è www.smelzo.it. Come sempre è a disposizione per ricevere suggerimenti o richieste sull'articolo all'indirizzo di posta elettronica francesco@smelzo.it

QUIZBOT: GIOCARE CON MESSENGER

LA POSSIBILITÀ DI INTERFACCIARSI A WINDOWS LIVE MESSENGER TRAMITE LA CREAZIONE DI ADDIN PERMETTE LA CREAZIONE DI SOLUZIONI ANCHE DIVERTENTI. IN QUESTO ARTICOLO CREEREMO UN "PASSAPAROLA" ELETTRONICO

Il nuovo Microsoft Live Messenger oltre ad una nutrita serie di funzionalità e ad una grafica più accattivante rispetto ai suoi diretti predecessori, porta con sé anche la possibilità di espandere le sue già ricche funzioni tramite la creazione di AddIn personalizzati. C'è ancora molta strada da fare in quanto ad oggi non è ancora disponibile un vero e proprio SDK che abiliti automaticamente la gestione degli AddIn da parte di Messenger o dei Template per Visual Studio che permettano di creare velocemente un nuovo progetto AddIn. In ogni caso abilitare gli AddIn ed utilizzare la classe per l'interfacciamento a Messenger non è un'operazione eccessivamente complicata. È necessario solo seguire alcuni semplici passi ed avere un minimo di padronanza degli strumenti di Windows.

In questo articolo vedremo come sviluppare un progetto che ci permetterà di conoscere a fondo questo argomento e allo stesso tempo creare un qualcosa che potrà farci divertire imparando. Vedremo inoltre molte delle possibilità offerte dalla libreria *Messenger Client.dll* nonché i suoi attuali limiti che verranno certamente superati con le sue future nuove versioni. Prima di cominciare, affinché si possa testare il progetto che ci accingiamo a sviluppare è indispensabile avere a propria disposizione almeno due account Messenger. Un account sarà utilizzato per l'esecuzione del codice che andremo a scrivere e fungerà da "bot", mentre l'altro o gli altri account saranno i "giocatori". Ecco nella seguente immagine una brevissima sessione di gioco:

```
KaOne scrive:
!ask
QuizBot scrive:
=< QUIZBOT VI DA IL BENVENUTO >=

Vi saranno poste 4 domande per ciascuna delle quali avrete a disposizione 30 secondi di tempo per rispondere.
Buona Fortuna e Buon Divertimento!
================================
Il gioco ha inizio...
===> Domanda n° 1: La capitale del Belgio? (vale [6] punti)
KaOne scrive:
Bruxelles
QuizBot scrive:
Bravo KaOne ora hai - 6 - punti.
===> Domanda n° 2: Il nome del nuovo sistema operativo di Microsoft? (vale [2] punti)
KaOne scrive:
Vista
QuizBot scrive:
Bravo KaOne ora hai - 8 - punti.
===> Domanda n° 3: Qual  il nome di Papa Ratzinger? (vale [0] punti)
KaOne scrive:
Josef
QuizBot scrive:
===> Tempo scaduto. La risposta era: Joseph

===> Domanda n° 4: Chi ha scritto "I promessi sposi"? (nome e cognome) (vale [5] punti)
KaOne scrive:
Alessandro Manzoni
QuizBot scrive:
Bravo KaOne hai guadagnato - 13 - punti.

Il gioco è terminato.
Ha vinto il giocatore:
KaOne (kaone@email.it)
con un totale di: 13 punti.
================================
Di seguito la classifica completa:
KaOne (kaone@email.it): 13

Blocca   Tipo carattere   - Emoticons
Invia
```

Figura 1: Esempio di sessione di gioco

LE REGOLE DEL GIOCO

Quello che andremo a sviluppare in questo articolo è un vero e proprio gioco a quiz del quale noi svilupperemo il "bot", ovvero un conduttore (una specie di Jerry Scotty), che proporrà delle domande agli utenti (giocatori) e valuterà le risposte da questi pervenute. Ad ogni domanda corrisponderà un punteggio che sarà accreditato ad ogni giocatore ogni qualvolta la risposta sarà esatta. Al termine della sessione di gioco, il bot stilerà una classifica con i nominativi di tutti i giocatori che hanno partecipato al gioco ed i loro rispettivi punteggi raggiunti.

Queste le regole principali:

- La sessione di gioco ha inizio quando un utente autorizzato invia al bot il comando: "!ask"
- QuizBot saluta i giocatori e propone la prima domanda specificando quanti punti vale
- Per rispondere alla domanda i giocatori hanno a disposizione un tempo predefinito oltre il quale: la domanda viene annullata, viene data la risposta corretta e si procede con la domanda successiva (in tal caso nessun giocatore si aggiudicherà il punteggio relativo)
- Se un giocatore fornisce la risposta corretta, viene comunicato il punteggio accumulato

fino a quel momento dal giocatore e si passa alla domanda successiva

- Se un giocatore fornisce una domanda errata viene incrementato un contatore in modo tale da riproporre la domanda ogni 5 risposte errate
- Il gioco termina al raggiungimento del numero massimo di domande per sessione o quando si esauriscono tutte le domande a disposizione di QuizBot
- Al termine di una sessione di gioco, viene visualizzato un riepilogo che comunica il giocatore vincitore con il punteggio raggiunto e a seguire la classifica di tutti gli altri utenti partecipanti con i loro rispettivi punteggi

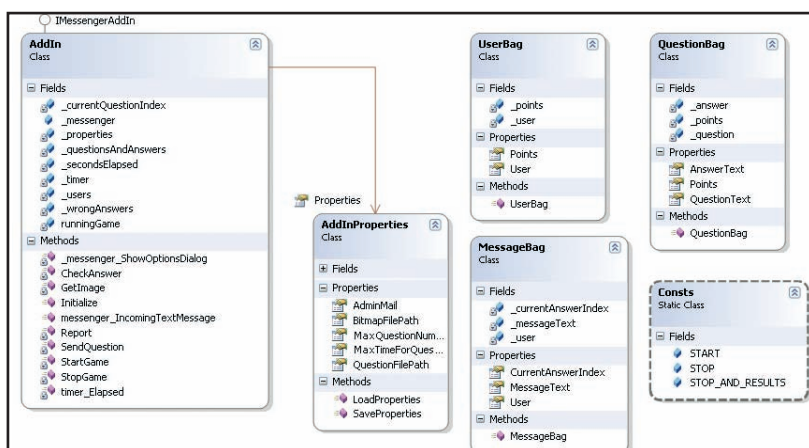



Figura 2: Class Diagram del progetto

IMPLEMENTAZIONE

Ora che conosciamo le regole del gioco vediamo come implementarlo. Elenchiamo prima di tutto le componenti principali del progetto:

- La classe contenente l'entry point dell'AddIn e la gran parte della logica di gioco
- Una classe Form per le impostazioni dell'AddIn
- Una classe serializzabile contenente le impostazioni dell'AddIn
- Una classe helper per interpretare i comandi
- Una classe bag per ognuna delle seguenti entità:
    - Messaggio
    - Domanda
    - Utente
- Una classe di costanti
- Il file di testo delle domande (questions.txt)
- Un file bitmap per rappresentare QuizBot in Messenger (bot_image.bmp)

Il progetto che svilupperemo è un progetto di tipo Class Library e affinchè la DLL da questo prodotta possa essere considerata da Messenger quale suo AddIn, è necessario che questa soddisfi due semplici requisiti:

- Il nome della DLL dovrà corrispondere al nome del Namespace del progetto unito al nome della classe principale, quindi nel nostro caso *MessengerBot.AddIn.dll*, dove MessengerBot è il Namespace mentre AddIn è il nome della classe principale
- La classe dovrà implementare l'interfaccia *IMessengerAddIn* che fortunatamente richiede l'implementazione di un unico semplice metodo: *Initialize*.

Prima di poter implementare l'interfaccia *IMessengerAddIn* è necessario naturalmente referenziare nel progetto la DLL *MessengerClient.dll* che possiamo trovare solitamente nella cartella:

C:\Programmi\MSN Messenger oppure C:\Program Files\MSN Messenger

Fatto questo, potremo allora aggiungere nella classe principale la seguente using:

```
using Microsoft.Messenger;
```

che ci permetterà di accedere al modello ad oggetti della DLL *MessengerClient.dll* senza dover specificare ogni volta tutto il namespace Microsoft.Messenger.

Il momento fondamentale per l'applicazione è l'arrivo di un nuovo messaggio rappresentato dall'evento *IncomingTextMessage*. La gestione dei messaggi in arrivo e la loro successiva validazione quali comandi o risposte si sarebbe potuta risolvere elegantemente attraverso l'utilizzo di un secondo Thread pre-

ABILITARE GLI ADDIN PER LIVE MESSENGER

1. Se Messenger è aperto è necessario chiuderlo prima di procedere
2. Dal menu Start, scegliere Esegui e digitare: regedit seguito dal tasto Invio.
3. Individuare la chiave di registro: HKEY_CURRENT_USER \ Software \ Microsoft \ MSNMessenger
4. Aggiungere un nuovo valore di tipo DWORD cliccando con il tasto destro del mouse sul nodo MSNMessenger ‡ New ‡ DWORD Value e dare al nuovo valore il nome di: AddInFeatureEnabled
5. Assegnare ad AddInFeatureEnabled il valore 1
6. Avviare Messenger, effettuare l'accesso e verificare che in Strumenti ‡ Opzioni ‡ Componenti aggiuntivi, si siano abilitate le voci per la registrazione degli AddIn.

posto allo scodamento di una coda di messaggi in arrivo. In questo modo arrivata una risposta corretta, le altre eventuali risposte inviate successivamente sarebbero potute essere scartate e si sarebbe potuta dare risposta inviando due messaggi: l'uno contenente la comunicazione di risposta esatta e l'altro contenente la successiva domanda.
Questa soluzione però non è praticabile a causa di una limitazione della libreria *MessengerClient.dll* che permette di inviare solo un messaggio per volta e solo in risposta ad un messaggio in arrivo. In pratica non è possibile inviare messaggi senza che questi siano risposte ad altri messaggi. Siamo quindi obbligati a verificare ogni messaggio in arrivo in maniera sincrona e rispondere allo stesso nel momento in cui arriva. Per lo stesso motivo non è possibile passare automaticamente alla domanda successiva allo scadere del tempo massimo di risposta se non dopo che, scaduto il tempo massimo, almeno un giocatore non abbia inviato un messaggio. Vediamo quindi passo passo le parti più importanti del codice.
Se il messaggio in arrivo è un comando ed è stato inviato dall'amministratore, viene interpretato in maniera opportuna attraverso una semplice *switch* che confronta il comando arrivato con quelli predefiniti. Per questo confronto si è utilizzata una classe statica, *Consts* contenente semplicemente quattro membri costanti di tipo *string*:

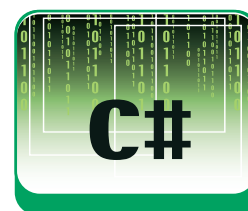
```
public static class Consts
{
   public const string START = "!ask";
   public const string STOP = "!stop";
   public const string STOP_AND_RESULTS = "!res";
   public const string PLUGIN_NAME = "QuizBot";
}
```

Nel metodo *messenger_IncomingTextMessage* andiamo quindi a confrontare il testo dei messaggi inviati per capire se si tratta di comandi:

```
public void messenger_IncomingTextMessage
 (object sender, IncomingTextMessageEventArgs e)
{
   try
   {
      // Verifica che il testo arrivato sia un
                                        comando valido
      if ( CommandHelpers.IsCommand
                                  ( e.TextMessage))
      {
         // Verifica che il comando sia stato inviato
```

```
                           dall'amministratore del gioco
         if (e.UserFrom.Email ==
                              Properties.AdminMail)
         {
            // Ricava il testo del comando
            string command =
CommandHelpers.ParseCommand(e.TextMessage);

            // Effettua il confronto con i comandi
                                                 noti
            if (command != null)
            {
               switch( command)
               {
```

Con l'istruzione *e.UserFrom.Email == Properties.AdminMail* ci assicuriamo che il comando sia stato inviato dall'utente autorizzato il cui indirizzo e-mail è stato specificato nella *OptionsDialog* di configurazione di QuizBot.
Se il comando inviato è di Start, viene avviato il gioco dopo un breve messaggio di benvenuto e viene posta la prima domanda:

```
case Consts.START:
  // Verifica che il gioco non sia già avviato
  if( !runningGame)
  {
      // Avvia il gioco e carica le domande
      StartGame();

      // Invia la prima domanda
      SendQuestion(_currentQuestionIndex
                        , "=< QUIZBOT VI DA IL
                                    BENVENUTO >="
                        + Environment.NewLine
```

LA CLASSE QUEUE

Il .NET Framework fornisce la classe Queue la quale permette di implementare una coda di tipo FIFO (First In First Out), ovvero il primo elemento ad entrare nella coda è il primo ad uscire. Questa classe è molto utile in tutti quei casi in cui si ha necessità di gestire una serie di oggetti in modo da rispettare il loro ordine di arrivo per l'elaborazione.

ABILITARE IL DEBUGGING DEGLI ADDIN

La possibilità di effettuare debugging del codice scritto per gli AddIn, deve essere abilitata esplicitamente attraverso i seguenti passi:

1. **Chiudere Live Messenger nel caso questo fosse aperto**
2. **Dal menu Start, scegliere Esegui e digitare: regedit seguito dal tasto Invio**
3. **Individuare la chiave di registro: HKEY_CURRENT_USER \ Software \ Microsoft \ MSNMessenger**
4. **Creare una nuova chiave AddInDebugHooks**
5. **All'interno della nuova chiave, creare un nuovo valore di tipo stringa: AddInDebugHooks e un nuovo valore di tipo DWORD: PromptForDebugger**
6. **Assegnare solo a PromptForDebugger il valore 1**

Nel momento in cui si andrà a caricare l'AddIn in Strumenti / Opzioni / Componenti aggiuntivi, apparirà una Message Box vuota contenente solo il tasto OK. Prima di premere OK, aprire il progetto dell'AddIn e agganciare Visual Studio al processo msnmsgr.exe dal menu Debug ‡ Attach to Process. Affinchè si possa eseguire il debug è necessario che il progetto sia stato correttamente compilato in modalità Debug.

```
                         + Environment.NewLine
                         + "Vi saranno poste "
                         +
     _questionsAndAnswers.Count.ToString()
                         + " domande per ciascuna
           delle quali avrete a disposizione "
                         +
   Properties.MaxTimeForQuestion.ToString()
                         + " secondi di tempo per
                                  rispondere."
                         + Environment.NewLine
                         + "Buona Fortuna e Buon
                              Divertimento!"
                         + Environment.NewLine
                         +
"==============================
                                            ="
                         + Environment.NewLine
                         + "Il gioco ha inizio..."
                         , e.UserFrom);   }
  else
  {
    _messenger.SendTextMessage("Il gioco è già
                        avviato", e.UserFrom);
  }

  break;
```

Se il comando è di Stop, viene interrotto il gioco senza fornire la classifica di gioco:

```
case Consts.STOP:
  // Se il gioco è in esecuzione, lo arresta
  if( runningGame)
  {
    StopGame(null, e.UserFrom, false);
  }
  break;
```

Se invece il comando è di Stop and Results, viene fermato il gioco e mostrata la classifica finale:

```
case Consts.STOP_AND_RESULTS:
// Se il gioco è in esecuzione, lo arresta e
                       visualizza la classifica
if( runningGame)
{
  StopGame(null, e.UserFrom, true);
}
break;
```

In questo ultimo pezzo di codice, la differenza tra la visualizzazione e la non visualizzazione della classifica è data dall'ultimo parametro del metodo StopGame che nel caso in cui sia true, invia la classifica utilizzando come destinatario del messaggio l'ultimo utente che ha inviato una risposta (esatta o errata che sia). Nel caso in cui invece il messaggio non è un comando allora viene considerato come risposta alla domanda correntemente attiva e come tale viene verificata la correttezza della risposta:

```
else
{
    CheckAnswer(new
            MessageBag(_currentQuestionIndex,
                 e.TextMessage, e.UserFrom));
}
```

Al metodo CheckAnswer passiamo una classe bag contenente l'indice della domanda corrente, il testo della risposta e l'utente che l'ha inviata. Il metodo a questo punto verifica la correttezza della risposta data:

```
if (question.AnswerText.ToLower() ==
                 message.MessageText.ToLower())
e nel caso in cui questa sia valida, incrementa il
                        punteggio dell'utente:
if (_users.ContainsKey(message.User.Email))
{
   (_users[message.User.Email] as
          UserBag).Points += question.Points;
}
else
{
   _users.Add(message.User.Email, new UserBag(
                  message.User, question.Points));
}
```

Successivamente risponde con il punteggio raggiunto fino a quel momento da quello specifico giocatore e propone la domanda successiva (nel caso ve ne fossero ancora altre) oppure conclude il gioco nel caso la risposta data fosse relativa all'ultima domanda della sessione di gioco:

```
if (_currentQuestionIndex <
                  _questionsAndAnswers.Count-1)
{
   // Incrementa il contatore delle domande
   _currentQuestionIndex++;
   // Azzera il contatore di risposte errate
   _wrongAnswers = 0;
   // Invia la domanda successiva
   SendQuestion(_currentQuestionIndex
                        , "Bravo "
                        +
                         message.User.FriendlyName
                        + " ora hai - "
                        +
                      (_users[message.User.Email] as
```

UTILIZZARE WEB MESSENGER

Web Messenger è la versione Web di Live Messenger. Offre funzioni limitate rispetto al suo fratello maggiore ma è molto utile quando non abbiamo il nostro computer a disposizione e abbiamo la necessità di utilizzare Messenger. E' sufficiente collegarsi all'indirizzo: http://webmessenger.msn.com e cliccare sul link "Accedi a MSN Web Messenger". Verrà richiesto di inserire l'indirizzo e-mail dell'account Messenger da utilizzare e la password. Dopo aver premuto il tasto "Accedi", si aprirà Web Messenger.

```
                                UserBag).Points.ToString()
                            + " - punti."
                            , userTo);
}
else
{
    StopGame( "Bravo "
                            +
                                message.User.FriendlyName
                            + " hai guadagnato - "
                            +
                        (_users[message.User.Email] as
                                UserBag).Points.ToString()
                            + " - punti."
                            , userTo, true);
}
```

Viene proposta la successiva domanda anche nel caso in cui fosse passato un lasso ti tempo superiore al limite preimpostato per dare la risposta e quindi i punti della domanda non vengono assegnati a nessun giocatore:

```
if (_secondsElapsed >=
                    Properties.MaxTimeForQuestion)
{
    string precAnswer =
((QuestionBag)_questionsAndAnswers.GetByIndex
            (_currentQuestionIndex)).AnswerText;
    if (_currentQuestionIndex <
                    _questionsAndAnswers.Count-1)
    {
        _currentQuestionIndex++;

        SendQuestion( _currentQuestionIndex
                            , "===> Tempo scaduto.
                                La risposta era: "
                            + precAnswer
                            + Environment.NewLine
                            , userTo);
    }
    else
    {
        // Viene inviata la risposta
        _messenger.SendTextMessage( "===>
                    Tempo scaduto. La risposta era: "
                                        +
                                precAnswer
                                        ,
                                    userTo);
        // Se siamo arrivati all'ultima domanda, viene
                                fermato il gioco
        StopGame( null, userTo, true);
    }
}
```

Se invece la risposta non fosse corretta, viene incrementato un contatore di risposte errate.

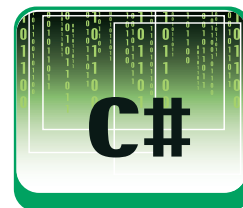
In questo modo è possibile dopo un certo numero di risposte errate, riproporre la domanda fatta in precedenza:

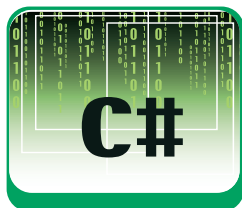
```
else
{
    // Incrementa il contatore delle risposte errate
    _wrongAnswers++;
    // Ogni 5 risposte errate, ripeto la domanda
    if (_wrongAnswers % 5 == 0)
    {
        SendQuestion(_currentQuestionIndex, null,
                                            userTo);
    }
}
```

Da notare che è stata usata come chiave univoca della hashtable degli utenti, il loro indirizzo e-mail, in quanto questa informazione è univoca per ogni account Messenger. Non è possibile cioè che due utenti distinti di Messenger si siano registrati con lo stesso indirizzo di e-mail.
Come si vede, inoltre, il censimento degli utenti avviene on-the-fly, ovvero gli utenti non vengono censiti preliminarmente ma soltanto la prima volta che inviano almeno una risposta esatta. Questo comporta l'esclusione automatica di tutti coloro che hanno totalizzato zero punti, lasciando in classifica solo chi ha totalizzato almeno un punto.
Non ci sono problemi a cambiare questo comportamento e a censire tutti gli utenti la prima volta che inviano una risposta anche errata, ma in questo modo si otterrebbe una classifica più prolissa e meno significativa perché potrebbero essere censiti utenti che invece di aver dato risposta ad una domanda hanno semplicemente scritto normali messaggi di chat senza partecipare al gioco.
Il tempo di risposta può essere gestito attraverso un semplice Timer con intervallo di un secondo:

```
private System.Timers.Timer _timer = new
                        System.Timers.Timer(1000);
```

Nell'evento *Elapsed* del Timer incrementeremo una nostra variabile globale di tipo *int* che ci indicherà in secondi il tempo trascorso dal momento in cui la domanda è stata posta ai giocatori:

```
private void timer_Elapsed(object sender,
                System.Timers.ElapsedEventArgs e)
{
    _secondsElapsed++;
}
```

LE CLASSI BAG

Le classi bag utilizzate in questo progetto sono semplici classi il cui unico scopo è quello di contenere le proprietà dell'entità a cui sono dedicate, facilitando in tal modo il passaggio di parametri nei metodi.

Sarà quindi sufficiente azzerare la variabile _secondsElapsed ad ogni nuova domanda per partire da zero con il conteggio del tempo trascorso. Passiamo ora al file di testo delle domande e delle risposte che presenta una coppia domanda / risposta per ogni riga, unitamente all'indice della domanda e al punteggio che questa permette di ottenere nel caso in cui vi si risponda correttamente. Le colonne sono delimitate dal carattere punto e virgola. Ecco un esempio di file di domande:

1;La capitale del Belgio?;Bruxelles;6
2;Il nome del nuovo sistema operativo di Microsoft?;Vista;2
3;Qual è il nome di Papa Ratzinger?;Joseph;8
4;Chi ha scritto "I promessi sposi"? (nome e cognome);Alessandro Manzoni;5

Nella prima colonna troviamo l'indice numerico univoco che identifica la riga, nella seconda colonna abbiamo la domanda, nella terza la risposta mentre per ultimo troviamo il punteggio che verrà assegnato all'utente che risponderà correttamente.
Il file delle domande può essere posizionato dove si desidera in quanto il suo percorso fa parte dei settaggi di QuizBot che è possibile impostare tramite la nostra dialog *OptionsDialog* a cui accederemo premendo il tasto "Impostazioni" nella finestra degli AddIn di Messenger

NOTA

REFERENZIARE DLL NEI PROGETTI

In Visual Studio per referenziare un assembly in un progetto è sufficiente cliccare con il tasto destro del mouse sul nome del progetto, scegliere la voce Add Reference... , cliccare sul tab Browse e selezionare dal disco rigido la DLL che s'intende referenziare nel progetto.

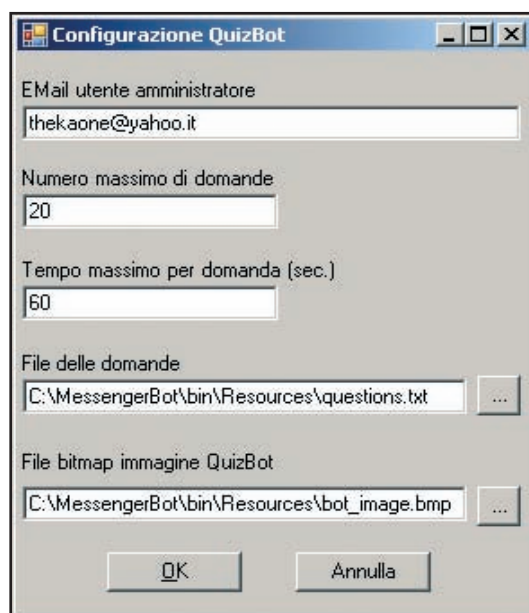



Figura 3: Impostazione delle opzioni di QuizBot

INSTALLAZIONE DI QUIZBOT

Dopo aver correttamente compilato la soluzione *MessengerBot* in modalità *Release*, troveremo l'assembly di QuizBot nella GAC. Si è reso necessario installare l'assembly nella GAC (Global Assembly Cache) in quanto questo è uno dei modi possibili affinché dagli AddIn di Messenger si possa accedere a risorse del computer locale, quali ad esempio file di testo, database, ecc. Visto che per il nostro progetto abbiamo utilizzato un file di testo per le domande ed un file immagine per rappresentare QuizBot, l'assembly deve essere necessariamente inserito in GAC. Affinché un assembly possa essere inserito in GAC, inoltre, è necessario che questo abbia uno *Strong Name* ovvero che sia stato creato un file chiave che lo legittimi ad essere inserito nella GAC di sistema. Prima di compilare il progetto e di registrare l'assembly in GAC è quindi necessario creare questo file con il comando (dal command prompt di Visual Studio):

```
sn -k messengerbot.snk
```

Questo comando genererà il file messengerbot.snk il quale dovrà poi essere copiato nella cartella del progetto e specificato nella proprietà di progetto Signing come mostrato in figura:

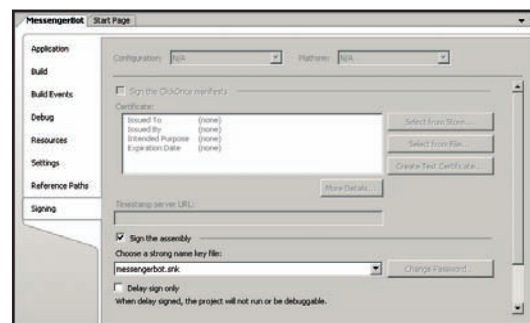



Figura 4: Impostazione file chiave

La registrazione in GAC viene svolta dal Post Build Event già impostato nel progetto che esegue la seguente riga di comando:

```
"C:\Program Files\Microsoft Visual Studio
  8\SDK\v2.0\Bin\gacutil.exe" /i "$(TargetPath)" /f
```

Nel caso in cui sul vostro PC il tool *gacutil.exe* non dovesse trovarsi nel percorso di cui sopra, sostituite il percorso corretto a quello già presente.

Per fare questo:

1. Cliccare con il tasto destro del mouse sul progetto nel Solution Explorer di Visual Studio
2. Scegliere "Properties"
3. Poi "Build Events" tra le voci presenti sulla

sinistra
4. Modificare il comando in "Post-build event command line"
5. Salvare e ricompilare il progetto

Se non si sono verificati errori, l'assembly è stato correttamente registrato nella GAC.
A questo punto avviare Messenger e:

1. Effettuare l'accesso con l'account messenger che si è deciso di utilizzare per QuizBot
2. Andare nel menu Strumenti Æ Opzioni Æ Componenti Aggiuntivi
3. Premere il tasto "Aggiungi a Messenger"
4. Selezionare l'assembly *MessengerBot. AddIn.dll* dalla cartella bin\release del progetto
5. Caricato correttamente l'AddIn, premere il tasto "Impostazioni"
6. Valorizzare opportunamente tutte le impostazioni di QuizBot:
   - Inserire nel campo "*EMail utente amministratore*" l'indirizzo e-mail del contatto Messenger che potrà inviare comandi a QuizBot
   - Specificare nel campo "*Numero massimo di domande*" il numero massimo di domande che si intende proporre ai giocatori
   - Nel campo "*Tempo massimo per domanda (sec.)*" specificare il tempo massimo in secondi concesso ai giocatori per rispondere a ciascuna domanda
   - Selezionare il file delle risposte
   - Selezionare il file bitmap che rappresenterà QuizBot in Messenger
7. Premere OK e nuovamente OK nella finestra principale

A questo punto per attivare QuizBot è sufficiente cliccare sul proprio nome account in Messenger per visualizzare il menu degli stati e selezionare la voce Attiva "QuizBot" come mostrato in **figura**. Da questo momento in poi QuizBot sarà in ascolto e risponderà a tutti i comandi inviati dall'utente amministratore. Inviando il comando "!ask" il gioco avrà inizio e gli utenti potranno cominciare a rispondere alle domande.

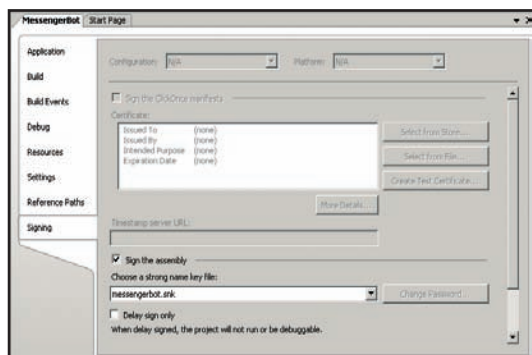



Figura 5: Assembly registrato nella GAC

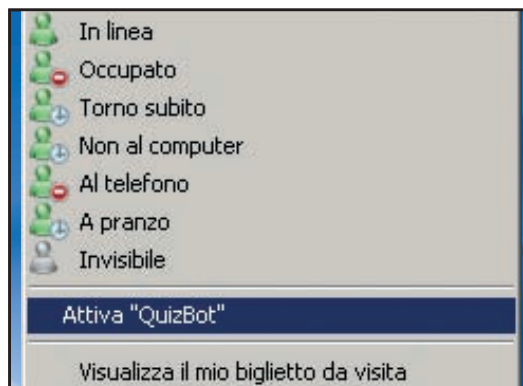
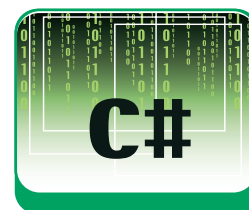



Figura 6: Attivazione di QuizBot nella finestra Stati di Messenger

GIOCHIAMO!

Come dicevamo nell'introduzione, per testare QuizBot sarà necessario avere a disposizione almeno due account Messenger. Uno da utilizzare come QuizBot e un altro per fare le veci di un giocatore. Visto che non è possibile avere due istanze di Messenger aperte con account diversi sullo stesso computer (almeno senza utilizzare software di terze parti), potete utilizzare l'account che farà da giocatore tramite Web Messenger. Dopo essersi collegati con l'account del giocatore e averlo anche definito come amministratore del gioco (attraverso la OptionsDialog a cui abbiamo accennato prima), potremo inviare il comando "!ask" per avviare la sessione di gioco. In qualsiasi momento durante la sessione di gioco sarà possibile inoltre inviare i comandi "!stop" per fermare immediatamente il gioco senza visualizzare la classifica oppure "!res" per fermare il gioco e visualizzare la classifica valida fino a quel momento.

CONCLUSIONI

Come abbiamo avuto modo di vedere in questo articolo, sebbene ci siano ancora alcune cose da migliorare, la libreria MessengerClient.dll offre un modello ad oggetti semplice ma che permette di creare applicazioni anche di una certa complessità. I possibili sviluppi di questo gioco potrebbero essere l'implementazione di nuovi comandi amministrativi, per gestire l'inserimento, la modifica e la cancellazione delle domande, la gestione di gruppi di domande, oppure la gestione di una classifica di gioco a lungo termine per permettere ai giocatori di mantenere il loro punteggio complessivo nel tempo. L'unico limite sarà costituito dalla nostra fantasia.

Gianni Malanga

GIANNI MALANGA

Lavora da più di 8 anni con tecnologie Microsoft in particolare nel campo dello sviluppo Web. Sviluppa in .NET ormai da alcuni anni e ha svolto diverse docenze per corsi di formazione professionali. Dopo aver lavorato alle dipendenze di importanti realtà locali, attualmente ha intrapreso la libera professione costituendo la ditta individuale Kaone Consulting che si occupa di consulenza informatica per PMI su tutto il territorio nazionale.
Gli si può scrivere all'indirizzo kaone@email.it

DELPHI CREA I SERVIZI WINDOWS

IN QUESTO ARTICOLO UNIREMO INSIEME TRE TECNOLOGIE DEL COMPILATORE DI BORLAND: ACCESSO AI DATABASE, PROTOCOLLO SMTP E WIN SERVICES PER COSTRUIRE UN SOFTWARE CHE CI CONSENTA DI INVIARE NEWSLETTER. VEDIAMO COME...

Il nostro intento è realizzare un completo servizio di gestione delle newsletter con Delphi e Firebird/Interbase embedded. Il nostro sistema si occuperà di controllare periodicamente un database contenente due tabelle, in una delle quali saranno conservati i testi da spedire, nell'altra gli utenti destinatari della newsletter, infine si occuperà fisicamente della spedizione. Una volta letto l'articolo saremo in grado di:

- Sviluppare un servizio windows
- Spedire una email con Delphi
- Gestire i dati di una query e di una tabella su un database Interbase o Firebird.

Per semplificare la parte di deployment, scegliamo la versione di Firebird 1.5 embedded. Potete scaricarla direttamente all'indirizzo http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_download_15 oppure utilizzare quella disponibile in questo stesso numero di ioProgrammo

PARTIAMO COL DB

Nel Db creeremo 2 tabelle, una che conterrà le news e una i destinatari. Nella tabella news. Il campo **INVIATA** sarà settato a **N** per quelle non inviate e a **S** per quelle inviate. In modo similare, nella tabella **DESTINATARI**, il campo **ATTIVO** indicherà se il destinatario deve ricevere le news o no (S/N). Qui di seguito le istruzioni SQL per generare le due tabelle:

```
CREATE TABLE LISTANEWS (
    ID          INTEGER NOT NULL,
    DESCRIZIONE  VARCHAR(50),
    DATA         DATE,
    TESTO        VARCHAR(1024),
    INVIATA      CHAR(1)
);
CREATE TABLE DESTINATARI (
    ID          INTEGER NOT NULL,
    NOMINATIVO  VARCHAR(50),
    MAIL        VARCHAR(50),
    ATTIVO      CHAR(1) DEFAULT 'S'
);
```

In aggiunta abbiamo anche creato due generatori **GEN_DESTINATARI_ID** e **GEN_DESTINATARI_ID** che hanno il compito di memorizzare l'ultimo ID inserito per i **DESTINATARI** e per le **NEWS**. Abbiamo infine creato due trigger: **DESTINATARI_BI** e **LISTANEWS_BI BEFORE INSERT** che si occupano di inserire il valore nei rispettivi ID e di incrementare il valore dei generatori.

E ADESSO DELPHI

Il progetto è formato da due eseguibili: il client per manipolare i dati e il servizio vero e proprio. Introduciamo innanzitutto alcuni componenti che andremo ad utilizzare:

TIBDatabase: è il componente incluso in Delphi per la connessione al DB. Facendo doppio click su di esso si apre una finestra che facilita il setup. In questa form si deve decidere se usare un Db locale (opzione obbligatoria per l'embedded) o server; nel caso del server va indicato server e protocollo. Poi utente,password, ruolo, charset e se si vuole il prompt per il login.
TIBTransaction: questo componente rappresenta la transazione. Normalmente si imposta a read-committed. Eseguita questa impostazione i 2 componenti vanno impostati reciprocamente come default Transaction e default Database.
TIBTable: è il componente che mappa una tabella del DB. Qui si deve impostare il Database e la Transazione, oltre che il nome della tabella.
TIBQuery: Analogo alla tabella, ma al posto del nome della tabella, qui va impostato il testo della query. Entrambe si aprono con Open.
TTimer: è un semplice componente che se abilitato esegue un metodo ad ogni passare dei millisecondi impostati sulla proprietà Interval.
TIdSMTP: è un componente della famiglia Indy Client che implementa il Server SMTP. Basta settare

alla proprietà Host l'indirizzo o il nome del mail server SMTP.

TIdMessage: è l'oggetto sempre degli Indy che rappresenta il messaggio email. Le proprietà da impostare sono: *From.Address, Recipients.EMail Addresses, MailMessage.Subject, Body.Text.*

PARTIAMO DAL CLIENT

Su una form posizioniamo un **TPageControl** dalla palette Win32 di Delphi. Poi aggiungiamo 2 pagine: **tsNews** e **tsDestinatari**. Nel primo tab mettiamo un **Tpanel** allineato al Top, una **TMemo** allineata al Bottom e una **TDBGrid** allineata al Client. Nel pannello aggiungiamo un **TDBNavigator**. Realizziamo il tab Destinatari come il precedente ma senza TMemo. Aggiungiamo poi al progetto un **TDatamodule** all'interno del quale metteremo tutti i componenti non visuali che interagiscono col DB: un **TIBDatabase**, **TIBTransaction**, due **TIBTable** e due **TDataSource**. Facendo doppio click su **IBDatabase1**, settiamo tutti i parametri della connessione al Db. Impostiamo reciprocamente transazione e connessione ai due componenti.

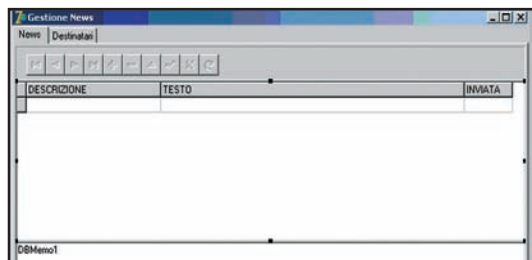



Figura 1: Come appare la form del client

Poi passiamo ad assegnare alle due tabelle **TIBTable** i parametri relativi a database, transazione e nome della tabella associata: **Destinatari** per la prima e **ListaNews** per la seconda. Fatto questo facciamo doppio click sui due componenti **TIBTable** per aprire la form dei Fields e quindi col tasto di destra rendiamo persistenti tutti i campi (Add all fields). Lo stesso facciamo con le **DBGrid** avendo poi l'accortezza di settare nei campi **INVIATA** e **ATTIVO** la

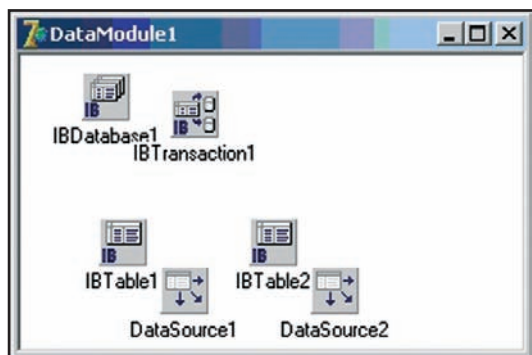



Figura 2: Il DataModule e i componenti

PickList così da far comparire una combo e le voci S o N.

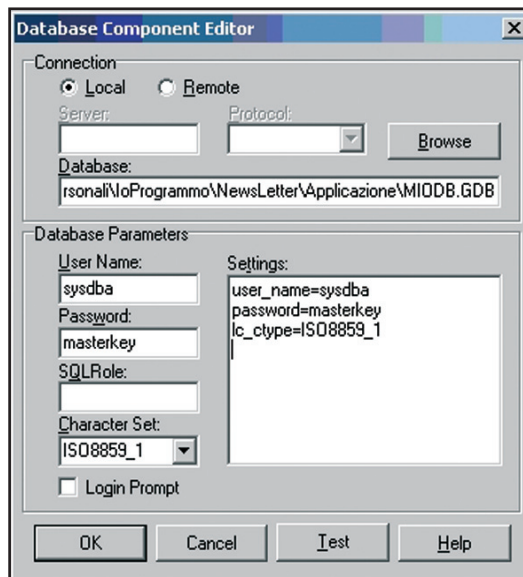



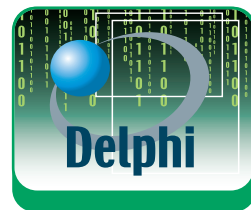
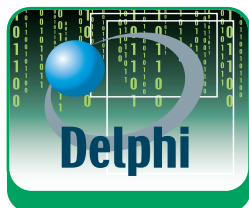
Figura 3: La form per settare le proprietà della connessione al DB

A questo punto la parte visuale è pressoché finita; iniziamo a lavorare sul codice.

IL METODO ONCREATE DEL DATAMODULE

Quello che vogliamo è che alla creazione del **TDataModule**, venga impostata la proprietà *DataBaseName* del componente **IBDatabase**. Il Db deve essere nella stessa directory dell'eseguibile, in questo modo usando la funzione *ExtractFilePath(Application.ExeName)* possiamo trovare il path sia dell'eseguibile sia del DB. Nell'esempio abbiamo usato una variabile globale (pathExe) per memorizzare il path.

FIREBIRD: UNA COSTOLA DI INTERBASE

Nell'esempio abbiamo usato Firebird 1.5 versione embedded, scaricabile dal sito www.ibphoenix.com/ (o http://www.firebirdsql.org/) Firebird è un database relazionale open source. Può girare su Windows, Linux e su molti Unix. Supporta quasi totalmente il linguaggio ANSI SQL 99, ammette trigger, stored procedure e relazioni. E' possibile anche fare in Delphi proprie funzioni (UDF) e registrarle in Firebird. Nasce dal codice di InterBase che è proprietario di CodeGear/Borland e che è un prodotto commerciale venduto e supportato direttamente da CodeGear. In questo esempio abbiamo usato la versione embedded, che richiede alcune piccole modifiche ad uno o più file presenti nel file scaricabile da internet. Quella essenziale è rinominare il client in fbclient.dll o Gds32.dll che sono i client "regolari" di Firebird e Interbase. Tutte le istruzioni sono comunque contenute nel file compresso che si scarica da sito.

```
procedure TfPrincipale.FormCreate(Sender: TObject);
begin
  pathExe := ExtractFilePath(Application.ExeName);
  if pathExe[Length(pathExe)] <> '\' then
    pathExe := pathExe + '\';
end;
```

Le uniche altre funzioni da implementare sono quelle legate all'OnShow dei due tab del PageControl, che apriranno le due tabelle associate con *DataModule1.IBTable1.Open* e *DataModule1 .IBTable2.Open.* Tutta la parte di gestione dei dati a questo punto e delegata ai componenti.

COSTRUIAMO IL SERVIZIO

Uno dei punti di forza di Delphi è quello di rendere semplice la creazione di una vasta serie di applicazioni/servizi. Per creare un servizio scegliamo dal menu file/New-Other la palette New e selezioniamo Service Application. A questo punto compare una form simile a un datamodule ma di tipo TService in cui andremo a posizionare tutti ciò che ci serve per implementare il servizio.

Sul componente TService posizioniamo un **TIBDatabase** e un **TIBTransaction** come nel datamodule del client e lo settiamo allo stesso modo. A questo punto andiamo ad aggiungere tre componenti **TIBQuery** che chiameremo: *iqNews*, *iqDestinatari* e *ibqUpdate* tutte assegnate al DB e alla transazione. La prima serve per selezionare le news non ancora inviate. Il testo SQL delle query è :

```
select DESCRIZIONE, ID, INVIATA, TESTO from
                    LISTANEWS where INVIATA = 'N'
```

iqDestinatari invece viene usata per ottenere gli indirizzi email dei destinatari; il testo della query è:

```
select ATTIVO, MAIL, NOMINATIVO from DESTINATARI
                                  where ATTIVO = 'S'
```

Infine con **ibqUpdate** andiamo ad aggiornare le news inviate; qui non mettiamo il testo della query, ma lo componiamo da codice, in alternativa avremmo anche potuto costruire una query parametrica e settare solo il parametro da codice.

Sull'evento *onCreate* del *TService* andiamo a leggere il path del DB e alcuni altri parametri che andremo subito a riscrivere nel file di INI.

```
procedure TService1.ServiceCreate(Sender: TObject);
var mioIni : TIniFile;
begin
  pathExe := ExtractFilePath(Application.ExeName);
  if pathExe[Length(pathExe)] <> '\' then
        pathExe := pathExe + '\';
  IBDatabase1.DatabaseName := pathExe +
                                       'MIODB.GDB';
  mioIni := TIniFile.Create(ChangeFileExt
                        (Application.ExeName,'.ini'));
  mailServer := mioIni.ReadString(
           'SETTING','MAIL_SERVER','mail.pippo.it');
  mioIni.WriteString(
              'SETTING','MAIL_SERVER',mailServer);
  daMail := mioIni.ReadString(
                       'SETTING','DA','pippo@tin.it');
  mioIni.WriteString('SETTING','DA',daMail);
  mioIni.Free;
  IdSMTP1.Host := mailServer;
end;
```

Questa modalità di operare ha il vantaggio di creare nel file INI una serie di voci che eventualmente l'utente potrà poi settare. Per gestire il file ini definiamo nel codice un oggetto mioIni di tipo TIniFile; per poter far questo dobbiamo aggiungere alla clausola uses la unit IniFiles, meglio se nella sezione implementation. Come per il DB il file INI deve essere posizionato nella stessa directory dell'eseguibile. Pertanto usiamo la funzione ChangeFileExt per ottenere il path completo del file:

```
mioIni := TIniFile.Create(ChangeFileExt(
                        Application.ExeName,'.ini'))
```

La funzione ReadString ad esempio legge un'impostazione dal file .ini se non ne trova una imposta quella di default. La logica è che al primo avvio del servizio verrà scritto il file .ini con le impostazioni di default. Se l'utente dovesse effettuare delle modifiche all'interno, automaticamente il servizio recuperebbe le nuove informazioni.

LA LOGICA DI BUSINESS

Nell'oggetto **TService** aggiungiamo 3 altri componenti: un **TTimer** dalla palette system, un **TIdMessage** e un **TIdSMTP** dalla palette Indy Clients. Nell'onCreate del **TService** andiamo anche ad assegnare l'host del IdSMTP1 a quello contenuto nella variabile mailserver letta dall'ini.

L'oggetto TTimer è quello in cui mettiamo tutta la logica. Per prima cosa settiamo la proprietà interval a 10000 millisecondi per fare un controllo ogni 10 secondi. Tale valore potrebbe essere portato a 60000 o più in produzione.

Andiamo ora ad implementare la funzione Timer1Timer che viene appunto "sparata" al passare di 10000 millisecondi.

```
procedure TService1.Timer1Timer(Sender: TObject);
```

```
var miaLista : TStringList;
  dataTimeS : string;
begin
  iqDestinatari.Open;
  iqNews.Open;
  miaLista := TStringList.Create;
  while not iqNews.eof do
    begin
      iqDestinatari.First;
      while not iqDestinatari.eof do
        begin
          miaLista.Add('Destinatario: ' +
                     iqDestinatariNOMINATIVO.AsString);
         miaLista.Add('Mail: ' + iqDestinatariMAIL.AsString);
         miaLista.Add('Oggetto: ' +
                       iqNewsDESCRIZIONE.AsString);
          miaLista.Add(iqNewsTESTO.AsString);
          MailMessage.Clear;
          MailMessage.From.Address := daMail;
          MailMessage.From.Name := daMail;
          MailMessage.Recipients.EMailAddresses :=
                     trim(iqDestinatariMAIL.AsString);
          MailMessage.CCList.Clear;
          MailMessage.BccList.Clear;
          MailMessage.Subject :=
                        iqNewsDESCRIZIONE.AsString;
         MailMessage.Body.Text := iqNewsTESTO.AsString;
          try
            if not IdSMTP1.Connected then
              IdSMTP1.Connect(1000);
            IdSMTP1.Send(MailMessage);
          except
            on e:exception do
              miaLista.Add(e.Message);
          end;
          iqDestinatari.next;
        end;
      ibqUpdate.SQL.Text := format('update LISTANEWS
                          set INVIATA = ''S'' where ID =
                            %d',[iqNewsID.asinteger]);
      ibqUpdate.ExecSQL;
      iqNews.next;
    end;
DateTimeToString(dataTimeS,'yyyymmddhhnnsszzz',now);
  miaLista.SaveToFile(pathExe + 'log'+
                               dataTimeS + '.txt');
  miaLista.Free;
  iqDestinatari.Close;
  iqNews.Close;
  IBDatabase1.Close;
  IdSMTP1.Disconnect;
end;
```

Qui il codice è abbastanza esplicativo, viene definita una variabile *miaLista* di tipo *TStringList*; questo oggetto è molto versatile, dobbiamo immaginarlo come una lista di stringhe. In questa variabile andremo a scrivere tutti i destinatari a cui inviare l la newsletter. A questo punto per ogni news non inviata andremo a controllare tutti i destinatari che dovranno ricevere la mail . La lista la useremo come contenitore delle informazioni di cui vogliamo tenere traccia. Quindi andremo ad aggiungere nominativo, mail, oggetto e testo del messaggio per ogni invio. A questo punto non resta che inviare la mail:

```
try
  if not IdSMTP1.Connected then
     IdSMTP1.Connect(1000);
  IdSMTP1.Send(MailMessage);

except
   on e:exception do
      miaLista.Add(e.Message);
end;
```

Dopo ogni invio eseguiamo:

```
ibqUpdate.SQL.Text := format('update
                  LISTANEWS set INVIATA = ''S'' where ID =
                            %d',[iqNewsID.asinteger]);
ibqUpdate.ExecSQL;
```

con cui andiamo a settare la news come inviata. Tentiamo la connessione al mail server settando un timeout di 1000 millisecondi; in caso l'invio fallisca, trascriveremo l'evento in un log file che poi salveremo:

```
DateTimeToString(dataTimeS,'yyyymmddhhnnsszz
                                        z',now);
miaLista.SaveToFile(pathExe + 'log'+dataTimeS + '.txt');
miaLista.Free;
```

In pratica con queste funzioni scriviamo in una stringa anno, mese,giorno,ora,minuti,secondi e centesimi in formato testo per generare ogni 10 secondi un file di log con nome diverso.

CONCLUSIONI

Uno dei punti di forza di Delphi è quello di occultare completamente una tecnologia al programmatore consentendogli di concentrarsi sulla logica di business dell'applicazione invece che sull'implementazione dell'inrastruttura di base. Nel nostro caso non abbiamo avuto alcuna necessità di conoscere il meccanismo di funzionamento dei servizi Windows così come del protocollo SMTP per poter realizzare un software che invece si basa appunto su queste due tecnologie. A margine annotiamo anche quanto sia stato semplice interagire con i database e con i file .ini

Regge Nicola

WINDOWS FORMS DATA BINDING

L'ITERAZIONE CON LE DIVERSE FONTI DI DATI COSTITUISCE UN PUNTO CRUCIALE DI OGNI APPLICAZIONE. PER QUESTO IL .NET FRAMEWORK FORNISCE DIVERSI STRUMENTI CHE ANALIZZEREMO DURANTE LO SVILUPPO DI UN PROGETTO REALE

La gestione dei dati è una delle parti fondamentali per la scrittura di una corretta applicazione. Ogni applicazione è fornita di una base dati che ne consente la memorizzazione. Il .NET Framework e di riflesso anche Visual Studio 2005 forniscono importanti strumenti per agevolarci nello sviluppo di quelle applicazioni che interagiscono con i database. Come connetterci al database? Come recuperare i dati e mostrarli sui nostri form in modo rapido ed efficace? Nel nostro articolo analizzeremo alcune sezioni di una ipotetica applicazione scritta per la gestione degli ordini di magazzino. In questa sezione creeremo un form per la visualizzazione e l'analisi degli ordini inseriti nella nostra applicazione sfruttando i meccanismi di binding, che letteralmente significa "legame", del .NET Framework.

IL DATABINDING IN WINDOWS FORMS

Il databinding in Windows Forms si è notevolmente evoluto nel passaggio alla versione 2.0 del framework. Visual Studio 2005, inoltre, offre un supporto molto avanzato per la fase di design. Il data binding è infatti possibile verso differenti tipologie di oggetti. DataSource, DataTable, array o collection possono essere legati direttamente o indirettamente ad un qualsiasi controllo che lo consente. Possiamo, inoltre, anche creare oggetti personalizzati, come le collection tipizzate, ed effettuarne il binding semplicemente implementando le adeguate interfacce. In .NET abbiamo due tipologie di binding: il Simple-Binding e il *Complex-Binding*. Il primo si realizza quando leghiamo la proprietà di un controllo ad una sorgente dati. Il secondo lo abbiamo legando la sorgente dati a controlli come il *DataGridView* o il *ListView*. Le due tipologie di binding vengono rappresentate in .NET attraverso due distinti manager:

- PropertyManager
- CurrencyManager

È possibile, in sostanza, utilizzare la classe *PropertyManager* per mantenere il legare tra controlli semplici e proprietà pubbliche di una classe. Questo manager non può essere utilizzato con controlli complessi come *ListBox*, *ComboBox* o *DataGrid*. A questo scopo viene utilizzata la classe *CurrencyManager*, con la quale è possibile effettuare il binding di controlli complessi verso qualsiasi tipo di oggetto che implementa l'interfaccia IList come un *DataGridView*, un *Array* o un *ArrayList*. Nel caso in cui volessimo creare un oggetto personalizzato e predisporlo per il binding possiamo, quindi, semplicemente implementare l'interfaccia *IList*.
Sia la classe *PropertyManager* sia la classe *CurrencyManager* ereditano da *BindingManagerBase*, che è gestita, a sua volta, dalla classe *BindingContext*. È importante sottolineare che ogni Windows Forms contiene almeno un oggetto *BindingContext* che, come anticipato, si occupa di gestire tutti i contesti di binding dei controlli associati e presenti sul form, stesso. Per inciso, è possibile creare anche più di un *BindingContext* per form. Per maggiori approfondimenti potete fare riferimento alla guida in linea del .NET Framework.
Lo scenario illustrato era quello maggiormente utilizzato nella versione 1.x del framework. Una novità importante del .NET Framework 2.0, infatti, è stata l'introduzione del *BindingSource*. Questo componente consente l'astrazione del livello di binding oltre alla possibilità di definire un maggiore controllo a design-time grazie all'integrazione più spinta con Visual Studio 2005.
Il *BindingSource* fa da intermediario tra il controllo e la classe sorgente, fornendo al contempo una serie di metodi che consentono di effettuare diverse operazioni come l'aggiunta di nuovi item, anche in modalità transazionale, la modifica o la rimozione di specifici oggetti, l'impostazione di filtri di ricerca oppure l'ordinamento degli oggetti gestiti. Per questi e per altri motivi l'uso dei **BindingSource** è quello preferito, da Visual Studio 2005 almeno, nella gestione dei bindings in Windows Forms. Bisogna prestare attenzione, però, al fatto che internamente questo oggetto fa un uso inten-

sivo di meccanismi di reflection, una caratteristica che penalizza le prestazioni in particolare su grosse mole di dati.

IL PROGETTO GEORD

Per meglio capire come funzionano i meccanismi di binding proveremo a realizzare un piccolo progetto che visualizzi un elenco di ordini. Nella nostra applicazione utilizzeremo il database AdventureWorks, fornito come esempio per Microsoft Sql Server 2005. Apriamo quindi Visual Studio 2005 e creiamo un nuovo progetto C# Windows Forms. Come riportato in figura 1, denominiamo il pro-

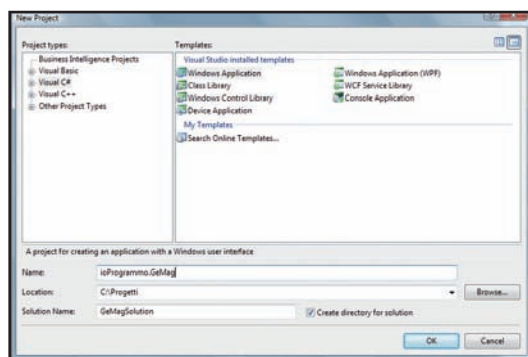



***Fig. 1:** Creiamo un nuovo progetto*

getto *ioProgrammo.GeOrd*, mentre chiamiamo la soluzione *GeOrdSolution*. Ovviamente il nome come il percorso in cui salvare il progetto è libero e di vostra scelta, ma nel corso di questo articolo utilizzeremo come riferimento i nomi sopra indicati.

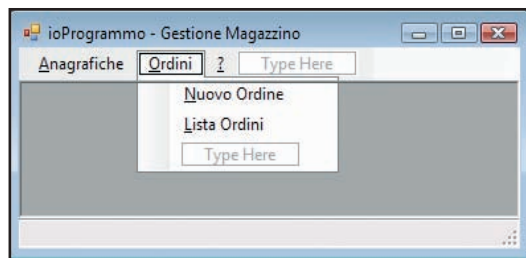



***Fig. 2:** Il controllo MenuStrip*

Nel progetto *GeOrd* creiamo un primo form con il nome *MainForm*. Questo form si occuperà di contenere tutti i form che compongono la nostra applicazione. Perciò impostiamo la proprietà *IsMdiContainer* del *MainForm* a true per indicare che il form conterrà altri form figli. Aggiungiamo al form un controllo *MenuStrip* per realizzare il menù principale con le voci *Anagrafica* e *Ordini*. Il form risultante sarà simile a quello in figura 2.
Aggiungiamo ora al progetto un nuovo form nel quale effettueremo la visualizzazione delle informazioni di testata in una prima griglia. Per ottenere questo risultato creiamo il form *ListaOrdiniForm* e lo aggiungiamo al progetto. Aggiungiamo al form un controllo DataGridView e lo chiamiamo *ordiniHeaderView*. Ora, sfruttando le caratteristiche di Visual Studio 2005, creeremo una connessione al database e visualizzeremo l'elenco degli ordini.

DOVE POSIZIONARE IL NOSTRO DATABASE

Durante l'installazione della nostra applicazione possiamo decidere dove installare il database. La scelta dipende dal tipo di applicazione che stiamo sviluppando e dalle modalità con cui deve essere eseguita. In caso di applicazioni standalone, il nostro database risiederà nella stessa posizione

VISUALIZZIAMO GLI ORDINI

Come già anticipato, nella nostra applicazione utilizzeremo il database AdventureWorks, liberamente scaricabile dal sito microsoft. Sul box laterale potrete trovare le informazioni per scaricare ed installare il database AdventureWorks. Per la realizzazione della nostra applicazione prenderemo in considerazione le tabelle *SalesOrderHeader* e *SalesOrderDetail*. Aggiungiamo ora il database come sorgente di dati del nostro progetto sfruttando gli strumenti messi a disposizione da Visual Studio 2005. Selezioniamo quindi il controllo *ordiniHeaderView* ed utilizzando gli smart tags creiamo un nuovo data source a livello di progetto, come visualizzato in figura 3.

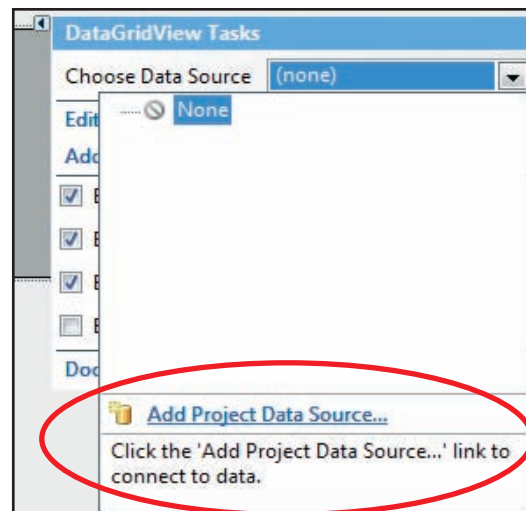



***Fig. 3:** Utilizziamo gli smart tag*

A questo punto Visual Studio 2005 ci mostra il Data Source Configuration Wizard che ci guiderà nella configurazione del data source. Nella scelta della tipologia di data source selezioniamo quindi Database e clicchiamo su *Next*, infine scegliamo la

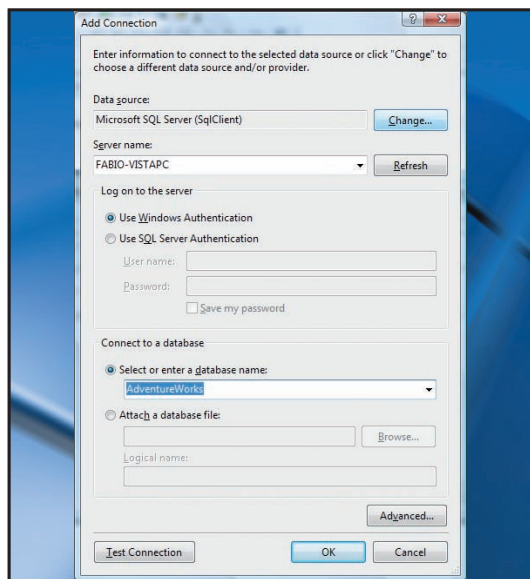



***Fig. 4:* Salviamo le informazioni nel file di configurazione**

connessione da utilizzare cliccando sul tasto New Connection. Impostiamo poi il data source su *Microsoft Sql Server* cliccando sul tasto *Change.* Sul campo Server Name selezioniamo il nome della macchina sulla quale è in esecuzione Sql Server 2005. A questo punto dovrebbe essere abilitato l'elenco dei database e qui selezioniamo il database *AdventureWorks*.
Nella successiva scheda scegliamo di salvare la connessione nel file di configurazione. Impostiamo infine le tabelle, definite nel precedente paragrafo, e clicchiamo su *Finish*. Con questa procedura Visual Studio 2005 crea un dataset ed un TableAdapter per ogni tabella aggiunta al progetto. Contemporaneamente ci chiede di configurare il *DataGridView* per selezionare il sorgente dei dati. Qui impostiamo come data source *SalesOrderHeader*, come visualizzato in figura 5, e deselezioniamo le caselle *Enable Adding*, *Enable Editing* e *Enable Deleting*.

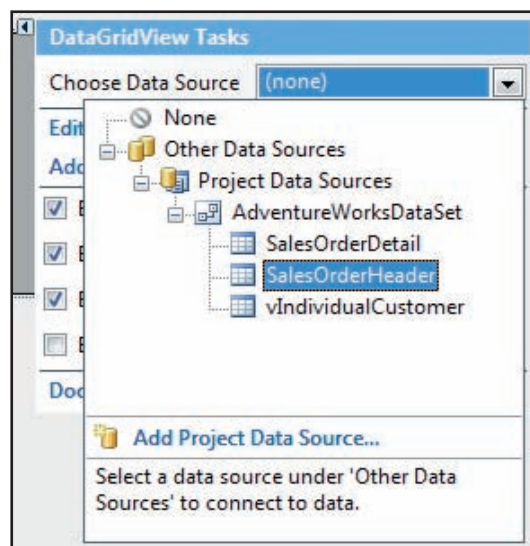



***Fig. 5:* Impostiamo il datasource**

Con questi passaggi viene creato un oggetto di tipo *BindingSource*, discusso nel precedente paragrafo, il cui nome è composto dal nome dell'oggetto, nel nostro caso la tabella *SalesOrderHeader*, associato al suffisso *BindingSource*, ottenendo nel nostro caso un istanza chiamata *salesOrderHeaderBindingSource*.
In questo modo abbiamo associato il *datasource*, o per meglio dire il *BindingSource*, direttamente al *DataGridView*.
Oltre alla procedura descritta, nella quale abbiamo visto come creare e contestualmente associare un data source ad un controllo, Visual Studio fornisce dei rapidi strumenti per la gestione di data source già configurati e correttamente aggiunti al progetto tramite il DataSource Explorer, accessibile dal menù *Data | Show Data Sources*.
Per ipotesi dobbiamo ora visualizzare il dettaglio degli ordini visualizzati nel DataGridView *ordiniHeaderView*. Qui possiamo visualizzare il dataset tipizzato creato prima e navigare anche nelle sue relazioni. Aprendo infatti la voce *SalesOrderHeader* vediamo tutte le proprietà/campi ed in più anche la tabella *SalesOrderDetail*,

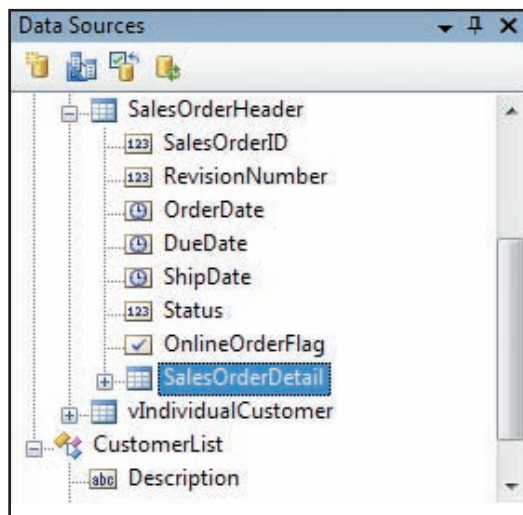



***Fig. 6:* Scegliamo la tabella di dettaglio**

navigabile appunto attraverso la relazione impostata nel dataset.
Come possiamo notare in figura 6, l'icona di fianco al nodo *SalesOrderDetail* rappresenta un DataGridView. Questo significa che semplicemente trascinando l'item sul form, verrà creato un DataGridView con tutte le impostazioni per risolvere la navigabilità master/details con il DataGridView *ordiniHeaderView*. Se proviamo ad effettuare l'operazione descritta e ad eseguire il progetto siamo in grado ora di scorrere le intestazioni degli ordini e di visualizzare il relativo dettaglio.

BINDING CON OGGETTI PERSONALIZZATI

Supponiamo ora di dover aggiungere nuove funzionalità al nostro form. Dobbiamo ora filtrare gli ordini per cliente e lo facciamo utilizzando dei business object da noi sviluppati. Aggiungiamo al progetto una semplice classe Customer, composta come di seguito, che conterrà il codice ed una descrizione:

```
class Customer
  {
      private string _idCustomer;
      private string _description;

      public string IdCustomer
      {
          get { return _idCustomer; }
          set { _idCustomer = value; }
      }

      public string Description
      {
          get { return _description; }
          set { _description = value; }
      }

  }
```

Contestualmente creiamo una collection che può essere esposta al *BindingSource*. Per farlo possiamo semplicemente implementare l'interfaccia *IList* o l'interfaccia *IBindingList*, purtroppo però questo non ci consentirebbe di utilizzare alcune caratteristiche avanzate del *BindingSource* come la possibilità di filtrare oppure la possibilità di ordinare i dati contenuti. Il framework fornisce già una implementazione dell'interfaccia *IBindingList* nella classe *BindingSource*, ma quest'ultima non implementa la possibilità di filtrare e ordinare i dati. Per poter utilizzare queste funzionalità dobbiamo crearci la nostra classe base, che estende la *BindingList* e che chiameremo *BindingListView*, ed infine tipizzarla nella nostra classe grazie all'implementazione dei generics.

Nel codice allegato alla rivista troverete la classe completa che per motivi di spazio non mostreremo.

Aggiungiamo quindi al progetto una nuova classe *CustomerList* che eredita da *BindingListView* e realizziamo ora un metodo statico che ci consentirà di recuperare i dati dal database attraverso una query e infine creare e restituire una nuova istanza della classe *CustomerList* con il risultato della query stessa. Il codice realizzato sarà infine simile a questo:

```
/// <summary>
/// Collezione di oggetti di tipo Customer.
/// </summary>
class CustomerList : BindingListView<Customer>
{
   /// <summary>
   /// Metodo statico per il recupero
                          delle informazioni dal database.
   /// </summary>
   /// <returns></returns>
   public static CustomerList GetCustomers()
   {
      // istanzio una nuova collection
      CustomerList customerList =
                                   new CustomerList();
      // creo una connessione al database
      SqlConnection connection =
                                 new SqlConnection();
      connection.ConnectionString =
   global::WinFormsDataBinding.Properties.Settings.
            Default.AdventureWorksConnectionString;
      // creo il comando e apro la connessione
                                            associata
      SqlCommand cmd = new SqlCommand
     ("SELECT CustomerID, FirstName, MiddleName,
        LastName FROM Sales.vIndividualCustomer",
                                        connection);

        connection.Open();
      // eseguo il comando e ciclo attravero
                                          il risultato
      SqlDataReader reader =
              cmd.ExecuteReader(CommandBehavior.
                                  CloseConnection);
      while (reader.Read())
      {
         // creo il singolo oggetto Customer
```

IL TABLEADAPTER

Il TableAdapter è una nuova classe introdotta con la versione 2.0 del .NET Framework. Consente di stabilire un canale di comunicazione tra l'applicazione ed il database. Essa, infatti, si occupa di mantenere le informazioni di connessione, di eseguire query e stored procedure e di ritornare i dati sotto forma di DataTable oppure di eseguire il fill dei dati stessi in un database esistente. Di default il TableAdapter implementa un metodo Fill ed un metodo GetData(), ma è possibile, utilizzando Visual Studio 2005 e il designer di DataSet tipizzati, creare i propri metodi. Riallacciandoci all'applicazione sviluppata nell'articolo, invece che filtrare il BindingSource potremmo recuperare l'elenco degli ordini per cliente creando un metodo FillByCustomerID, e il relativo GetDataByCustomerID, che recupera le informazioni sulla base di un parametro che indica l'ID richiesto. Possiamo quindi creare, modificare e rimuovere determinate query parametrizzate ed associarle a più di un metodo del relativo TableAdapter. Maggiori informazioni ed approfondimenti potete trovarle nella guida in linea o all'indirizzo http://msdn2.microsoft.com/it-it/library/bz9tthwx(VS.80).aspx

```
                e formatto il nome che sarà visualizzato
            Customer customer = new Customer();
            customer.IdCustomer = reader
                        ["CustomerID"].ToString();
            customer.Description = String.Format("{0},
            {1}", reader["LastName"].ToString(), reader
                        ["FirstName"].ToString());
            customerList.Add(customer);
        }
        // chiudo il datareader e la connessione
        reader.Close();
        // ritorno la lista di oggetti Customer
        return customerList;
    }
}
```

Ora ritorniamo sul form e apriamo il *Data Source Explorer*. Qui clicchiamo sul tasto *Add New Data Source*, nel wizard selezioniamo **Object** e clicchiamo su **Next**. Il wizard ci mostra un'elenco delle classi e dei dataset associati al progetto, divisi per namespaces. Qui selezioniamo la classe *CustomerList* e clicchiamo su **Finish**. Ora Visual Studio mostrerà anche la *CustomerList* nella *Data Source Explorer*. Selezioniamo il tipo di controllo che deve essere generato per quello specifico data source cliccando sulla relativa combo box, come mostrato in figura 7.

Una volta selezionata la generazione come controllo ComboBox, trasciniamo il nodo sul form. Visual Studio genera, come precedentemente fatto per la tabella *SalesOrderDetail*, un'istanza del *BindingSource* e contestualmente un'istanza del *BindingNavigator*, meglio descritto nel box laterale e che ora possiamo anche rimuovere. Impostiamo, quindi, le proprietà *DisplayMember* della ComboBox sulla proprietà *Description*, e la proprietà *ValueMember* sulla proprietà *IdCustomer*. Nell'evento *Load* del form *ListaOrdiniForm* inseriamo il seguente codice:

```
this.customerListBindingSource.DataSource =
                        CustomerList.GetCustomers();
this.customerListBindingSource.Sort = "Description";
```

Nella prima riga viene impostato il risultato del metodo *GetCustomer* come sorgente del *BindingSource*. Successivamente viene impostata la proprietà *Description* come proprietà per l'ordinamento dei dati. L'impostazione della proprietà per il sorting deve essere attentamente valutata, poichè il suo utilizzo potrebbe generare un decadimento delle prestazioni in quanto ogni volta viene ricreata l'intera lista sulla base dell'ordinamento impostato. Provando ad eseguire il progetto vedremo popolata la nostra ComboBox con i nomi dei clienti inseriti nel database. Per filtrare i risultati degli ordini sulla base del cliente selezionato, utilizziamo l'evento *SelectedIndexChanged* della ComboBox. Perciò nel relativo gestore dell'evento inseriamo il codice seguente:

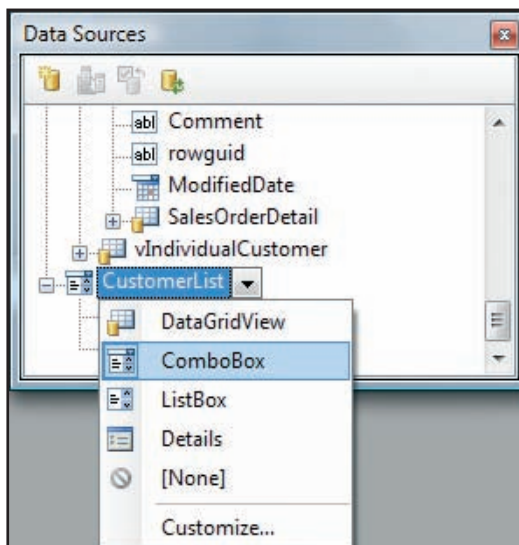



***Fig. 7:* Il tipo di controllo da generare**

```
this.salesOrderHeaderBindingSource.Filter =
        CustomerID = " + this.customerListComboBox.
                                    SelectedValue;
```

Semplicemente impostando la proprietà *Filter* del *BindingSource*, vengono filtrati i risultati e di conseguenza aggiornato il DataGridView. Notiamo che se allo stesso *BindingSource* associamo più di un controllo, ogni variazione verrà immediatamente replicata su tutti i controlli ad esso legato.

CONCLUSIONI

Nell'articolo abbiamo visto come è possibile utilizzare il databinding in Windows Forms e sfruttare le potenzialità e le agevolazioni fornite da Visual Studio. Abbiamo visto come effettuare il binding utilizzando sia un DataSet che un oggetto personalizzato appositamente costruito. La scelta nell'uso di un dataset o di un oggetto personalizzato dipende molto dal tipo di applicazione che stiamo sviluppando, ma è importante sapere che il supporto al binding è pressocchè identico indipendentemente dalla scelta fatta.

In modo solo leggermente si potrebbe usare come fonte di dati un web services oppure un formato personalizzato.

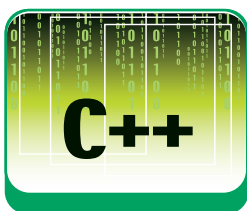
Per qualsiasi chiarimento potete contattarmi attraverso il mio blog, segnalato nel box laterale, oppure utilizzando il forum di ioProgrammo all'indirizzo http://forum.ioprogrammo.it.

Fabio Cozzolino

L'AUTORE

Fabio Cozzolino lavora come responsabile tecnico nello sviluppo di progetti web ed applicazioni distribuite presso la Fimesan Spa di Molfetta (BA). È cofondatore di DotNetSide, lo user group del sud italia il cui intento è quello di organizzare eventi di maggior spessore tecnico legati allo sviluppo con il .NET Framework. Possiede la certificazione MCAD.NET ed il suo blog è raggiungibile all'indirizzo http://www.dotnetside.org/blogs/fabio

UN GARBAGE COLLECTOR PER C++

ULTIMA PUNTATA SUL MEMORY MANAGEMENT. VEDREMO COME TRASFORMARE IL C++ IN UN LINGUAGGIO GARBAGE COLLECTED GRAZIE AL GC DI HANS BOEHM, E QUALI CAMBIAMENTI COMPORTA L'ADOZIONE DI QUESTO MODELLO DI PROGRAMMAZIONE.

In questa breve serie dedicata al memory management abbiamo affrontato diversi argomenti: dagli smart pointers, all'analisi del funzionamento "nascosto" degli operatori globali ::new e ::delete, fino agli allocatori STL.
Abbiamo così visto come il C++ permette di andare molto a basso livello nella gestione della memoria, arrivando a gestire i singoli byte nel free store e la modalità di allocazione, tanto che ne abbiamo approfittato per divertirci a scrivere un rudimentale leak detector.
Ma abbiamo anche visto che, attraverso l'uso degli smart pointers come auto_ptr e shared_ptr, e in generale dell'idioma RAII, è possibile far gestire la memoria e le risorse automaticamente alla macchina. Questo è sicuamente un approccio vincente, dal momento che l'errore umano è sempre in agguato, e il flusso dei progammi medio-grandi può essere veramente imprevedibile e inestricabile (per non parlare delle eccezioni!).
Chi è abituato a linguaggi di alto livello popolari come Java o C#, o anche meno "alla moda" come Smalltalk o Eiffel, vede perfino la gestione con gli smart pointers come una complessità inaccettabile. Le obiezioni tipiche sono di questo tenore: "E io dovrei ogni volta ricordarmi di "affidare" le mie strutture ad uno smart pointer? Dovrei vivere nel terrore di evitare ogni possibile riferimento circolare? E a che serve, poi, se uso librerie di terze parti scritte da gente che non sa gestire la memoria?".
Molte di queste (e altre) obiezioni sono indubbiamente di parte, segno di una scarsa volontà di adattarsi ad un differente sistema di programmazione; tuttavia alcune sono oneste, e tutte si fondano su una verità innegabile: nei linguaggi più "semplici" la gestione della memoria si presenta come un'attività molto più comoda e lineare.
Il meccanismo che permette tanta semplicità prende il nome di **Garbage Collector**: in quest'articolo vedremo come funziona un tipico GC, i pro e contro di questo sistema, la posizione del C++ a riguardo, e come integrare uno dei migliori e più portabili (nonché gratuiti) garbage collector all'interno dei nostri progetti.

COS'È UN GARBAGE COLLECTOR?

Un garbage collector è un meccanismo che si occupa di gestire il rilascio della memoria dinamica automaticamente. Si tratta di un'entità dall'apparenza "magica e intelligente", capace di osservare il free store "dall'esterno", e analizzare il flusso dell'esecuzione in modo da stabilire quando una risorsa non serva effettivamente più e possa quindi essere rilasciata con sicurezza.
Ciò è completamente differente dall'uso degli smart pointer, dal momento che non è connesso con lo srotolamento dello stack, né con qualsiasi altro evento deterministico. Un garbage collector può "decidere" di attivarsi quando ritiene che ce ne sia effettivamente bisogno: se la memoria scarseggia, o se il programmatore ha richiesto esplicitamente di fare un po' di pulizia. Allora si mette in moto, e dealloca tutte quelle risorse che sono ormai diventate irraggiungibili.
Tutto ciò libera il programmatore finale da una larga parte delle sue preoccupazioni abituali: scrivere istruzioni free/delete.

```
int main() {
    for (int i=0; i<10000; i++)
        new int;
}
```

Un codice come quello scritto qui sopra ha due difetti: non serve a niente, e provoca un memory leak piuttosto ingente, di almeno 20000 byte. Con un Garbage Collector il codice continua a rimanere insensato, ma quantomeno il leak viene recuperato, e la spazzatura ripulita. Ma in che modo? Nella computer science niente avviene per magia, ed è sempre bene essere al corrente di ciò che avviene dietro le scene. Nei paragrafi che seguono vedremo quindi come opera un garbage collector. Questo ci eviterà di cadere in luoghi (fin troppo!) comuni completamente infondati, e ci permetterà di sapere cosa possiamo realmente aspettarci da un simile strumento – e come facilitargli il lavoro.

COME FUNZIONA UN GARBAGE COLLECTOR?

Alla base del funzionamento del Garbage Collector sta l'abilità di capire quando una risorsa sia ormai diventata ***irraggiungibile***. Con questo termine si indica una qualsiasi risorsa che non sia più referenziata da alcun puntatore in vita (ovvero: "il cui use_count sia 0", se avete seguito i nostri precedenti appuntamenti).

Il Garbage Collector svolge il suo lavoro "per esclusione": al posto di individuare le risorse irraggiungibili, analizza tutto il free store stilando l'elenco di tutte quelle risorse sicuramente raggiungibili. Una volta ottenuto questo insieme, tutte le aree rimaste sono da considerarsi irraggiungibili e possono quindi essere eliminate.

Il primo passo consiste nello stilare l'elenco di tutte quelle risorse alle quali fanno riferimento diretto i puntatori attualmente visibili. Questi puntatori costituiscono pertanto la base (o, in gergo, radice) dell'analisi del garbage collector, e possono trovarsi, al momento della chiamata: nello stack, nell'area globale/statica, e nei registri.

```
//puntatore radice, nell'area globale
str* pGlobale;
int main() {
        //puntatore radice, nello stack
        string* pLocale = new string("Roberto");
        pGlobale = new string("Giabim");
        pLocale = 0;
        GC_gcollect(); //invochiamo il garbage
                                                collector
}
```

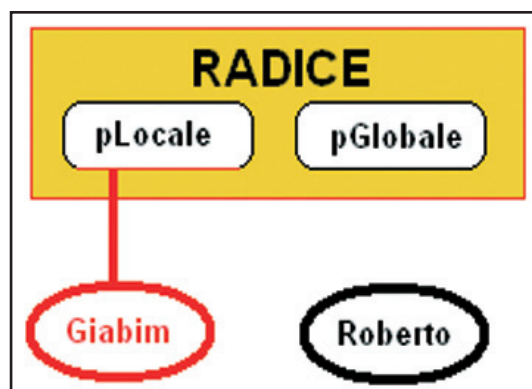



Figura 1: La fase di marcatura: l'oggetto in rosso è puntato dalla radice e sarà mantenuto, il nero è "solo" e sarà deallocato.

Nella situazione illustrata qui sopra, ad esempio, il Garbage Collector inizierà la sua analisi prendendo nota dei puntatori presenti nello stack (pLocale) e nell'area statica (pGlobale), e marchierà come raggiungibili le risorse a cui questi fanno riferimento. In questo modo non avrà problemi a capire che la stringa "Giabim" non deve essere distrutta, perché tenuta in vita da pGlobale. Le risorse rimanenti (ovvero la stringa "Roberto") possono quindi essere deallocate.

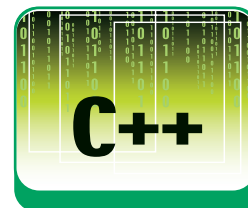
Detta così, la cosa sembra semplice. In realtà, come vedremo, il meccanismo è decisamente più complesso. Innanzitutto un GC non può tener conto soltanto dell'insieme radice, e quest'esempio mostra perché:

```
struct Eroe
{
        string nome;
        Eroe* figlio;
        //... costruttore, etc...
};

int main() {
        Eroe* eroe = new Eroe("Priamo");
        eroe->figlio = new Eroe("Ettore");
        eroe->figlio->figlio = new Eroe("Astianatte");

        GC_gcollect(); //invochiamo il garbage
                                                collector

        cout << eroe->figlio; //crash?
}
```

Seguiamo il garbage collector nella sua analisi. Cominciamo dalla memoria statica e globale: non c'è nulla. Nello stack, invece, c'è un puntatore, che punta alla risorsa *Eroe("Priamo")*: la salviamo (marcandola come "raggiungibile", o copiandola in un'altra area di memoria – a seconda della nostra politica). Nel free store si trovano altri due oggetti: li buttiamo?

Ovviamente, un simile comportamento ucciderebbe all'istante sia Ettore sia suo Figlio, nonché la nostra applicazione al momento dell'istruzione di inserimento in cout.

È evidente che, una volta definito l'insieme radice, occorre mantenere in vita anche tutti gli oggetti puntati dagli oggetti puntati, fino a percorrerne ricorsivamente tutta la gerarchia. Come mostra la figura 2, questo sistema funziona anche in presenza di riferimenti circolari.

Una volta che l'insieme degli irraggiungibili è stato

IL BDW-GC E LE MACRO

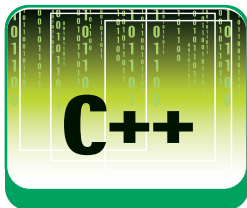
Molte delle cosiddette "funzioni" che fanno parte dell'interfaccia C del garbage collector sono in realtà delle macro, che si espandono in modo differente a seconda delle impostazioni d'ambiente. GC_MALLOC, ad esempio, richiama la funzione GC_malloc in condizioni normali, e GC_malloc_debug nel caso in cui GC_DEBUG sia stato definito. La maggior parte delle macro del garbage collector segue il medesimo schema.

individuato, poi, si può decidere che farne. Rilasciare la memoria? Riscriverci sopra nelle alloca-

Figura 2: Fase di marcatura di oggetti collegati fra loro. Anche se "Achille" e "Peleo" si puntano fra loro, non hanno alcun collegamento con la radice, e quindi vengono deallocati.

zioni future? Separarla dalla memoria raggiungibile, in modo da ricompattare il tutto e aumentare la località dei riferimenti? Un garbage collector viene caratterizzato proprio dalle politiche che assume in queste e altre scelte.

NOTA

GC_INIT()

Nel main del programma, prima di usare il collector, potrebbe essere necessario inizializzare il GC tramite questa macro. In molti compilatori/piattaform e, comunque, GC_INIT() si espande "a vuoto", e pertanto una chiamata "non fa né bene, né male".

PERCHÉ IL C++ NON HA UN GARBAGE COLLECTOR STANDARD?

Lo standard del C++ non fa alcun riferimento ad un possibile "Garbage Collector", e le questioni al riguardo riempiono spesso le pagine dei newsgroup e le bocche dei detrattori, che prendono l'occasione per definire il C++ come un "linguaggio vecchio, non aperto all'innovazione". Il fatto che le tecniche di garbage collection siano note e praticate dalla preistoria informatica dei primi anni sessanta dovrebbe far capire che:

a) Le polemiche sul "vecchiume" sono (nei migliori dei casi) uno dei tanti frutti ridicoli della confusione generata dal marketing IT.
b) La garbage collection è una tecnica consolidata e considerata utile da tanto tempo

E allora perché il C++ standard non ha un garbage collector? Verrebbero in mente tre motivi:

a) **La compatibilità**: non si può pretendere di buttare tutto il codice gestito manualmente finora.
b) **L'efficienza**: la gestione manuale delle risorse (se ben fatta) è mediamente molto più rapida e precisa di quella eseguita dal garbage collector.
c) **Il cambio di paradigma**: come vedremo, programmare con un garbage collector alle spalle implica degli stravolgimenti del flusso di esecuzione, soprattutto per quanto riguarda la finalizzazione delle risorse (file, thread, etc…) che non può più essere gestita in maniera deterministica.

Ma tutte queste argomentazioni sono solo apparenti, e si risolvono rendendo il garbage collector una possibilità opzionale, attivabile con direttive del compilatore o con l'inclusione di un header – in pieno accordo con il sacrosanto principio che ha guidato il C++ fin dagli albori: "non si deve pagare ciò che non si usa".
A testimonianza del fatto che non si tratta di un rifiuto ideologico, Bjarne Stroustrup ha confermato più volte che: "*quando (non "se") la garbage collection sarà disponibile, avremo due modi di programmare in C++*", aggiungendo peraltro di essere molto più favorevole all'adozione di un GC opzionale, piuttosto che all'introduzione degli smart pointer.
Molto probabilmente, dietro alla mancanza di un GC nel C++ standard c'è un motivo decisamente più pragmatico: le caratteristiche del linguaggio rendono molto ardua l'impresa di scriverne uno universale. L'aritmetica e reinterpretazione dei puntatori, i meccanismi di cast, la possibilità che (grazie all'ereditarietà multipla o ai puntatori interni) lo stesso oggetto corrisponda a più indirizzi diversi, le molte maniere in cui si può allocare un dato, sono tutti scogli insidiosi che ostacolano la stesura di specifiche standard che dovrebbero essere seguite da tutti i garbage collector C++.

IL GARBAGE COLLECTOR BDW

Tutte queste difficoltà non hanno scoraggiato le software house e i singoli individui dall'imbarcarsi nell'avventura di scrivere un garbage collector per C++.
La strada più agibile (e, quindi, più percorsa) è quella di **limitarsi ad una singola piattaforma**, e ad un singolo compilatore - spesso "il proprio". Così facendo, oltre ad eliminare le necessità di porting e compatibilità, è possibile ricevere dei grossi aiuti dallo specifico sistema operativo (o del framework) su cui ci si appoggia. Un caso emblematico è C++.NET, che con qualche estensione del linguaggio riesce ad usare agevolmente il garbage collector gestito dal

DIRETTIVE DI COMPILAZIONE

Uno dei punti di forza del BDW-GC sono le innumerevoli direttive di compilazione e variabili d'ambiente, che permettono di condizionare e ottimizzare il lavoro del garbage collector. Un'analisi approfondita andrebbe oltre le possibilità di quest'articolo, tuttavia potreste voler rimuovere dal makefile eventuali –DSILENT e –DNO_DEBUG, in modo da far mantenere al GC un utile registro con tutte le informazioni circa le allocazioni fatte.

framework sottostante.
Più difficile (e meno battuta) è invece la via di **realizzare un garbage collector "universale"** usando esclusivamente il C++ standard. Per tutte le ragioni che ho appena elencato (più infinite altre: allineamento, gestione dello heap, etc…), questo è un sogno impossibile, quantomeno allo stato attuale del linguaggio.
Hans Boehm, Alan Demers e Mark Weiser, sono comunque riusciti nel corso degli ultimi vent'anni ad andarci vicino, costruendo un garbage collector facilmente portabile, che riesce ad operare sulla maggior parte delle piattaforme e dei principali compilatori sul mercato.
Ciò che consente al garbage collector di fare un ottimo lavoro nonostante le difficoltà derivate dal linguaggio è il comportamento **conservativo**. Un garbage collector conservativo tenta di "indovinare" quali dati memorizzati in memoria sono puntatori, scansionando la memoria con euristiche appropriate. L'analisi fondamentale consiste nel vedere se è possibile rintracciare all'interno di un blocco di memoria una sequenza numerica che punti ad un'altra zona allocata: in questo caso il garbage collector assume sempre di avere a che fare con un puntatore. Il sistema, ovviamente, non è perfetto: è possibile che il garbage collector scambi, per pura coincidenza, delle zone di memoria "innocue" per puntatori, finendo col trattenere della memoria che potrebbe invece essere liberata, ma (rari casi a parte) ciò non causa alcuno spreco sensibile di risorse.
Il garbage collector BDW è stato usato in questi anni per innumerevoli progetti, spesso legati alla creazione di linguaggi di alto livello capaci, grazie ad esso, di compilare in un C magicamente "gestito". Gli esempi più noti comprendono il **compilatore GNU per Static Java**, e la versione per Linux del .NET Framework: Mono, oltre a decine di altri compilatori, interpreti e ambienti runtime.

INSTALLAZIONE DEL GARBAGE COLLECTOR

Usare il garbage collector BDW è molto semplice, ma installarlo può essere, in alcuni casi, molto meno piacevole. Se riuscite a superare lo scoglio dell'installazione tutto il resto sarà in discesa. Promesso.
Se usate Linux e GCC non avrete problemi. In tal caso l'installazione segue la procedura classica, con una piccola variante opzionale, data dalla possibilità di verificare la costruzione:

```
./configure --disable-threads
make
make check
make install
```

Sotto Windows, invece, le cose possono farsi un po' più complicate. Invece di prevedere ogni possibile soluzione – cosa che occuperebbe il resto dell'articolo – vi indicherò una serie di passi da compiere per riuscire ad usare il garbage collector in modo efficace – e soprattutto gratuito. Per gli indirizzi web delle applicazioni, fate riferimento al box laterale "Riferimenti per l'installazione".

1. Scaricate l'ultima versione del BDW Garbage Collector.
2. Decomprimetela dove volete (io assumo "C:\gc")
3. Scaricate e installate il Platform SDK.
4. Scaricate e installate Visual C++ 2005 Express.
5. Entrate nel prompt dei comandi.
6. Andate nella directory d'installazione del Platform SDK. Ad esempio:

```
CD C:\Programmi\Microso~1\
```

7. Eseguite SetEnv.cmd. In questo modo attiverete le variabili d'ambiente.

```
SetEnv
```

8. Andate nella directory del Garbage Collector:

```
CD C:\Gc
```

9. Eseguite il makefile che desiderate.

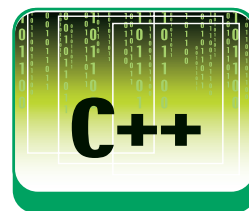
```
Nmake –f NT_MAKEFILE
```

A questo punto, se tutto va bene, il linker riuscirà a costruire il file C:\GC\gc.lib (mentre potrebbe fallire la compilazione dei test. Non fa niente; anzi, è "normale").
Ora potete creare con Visual Studio un nuovo progetto console, impostandone le proprietà in modo che:

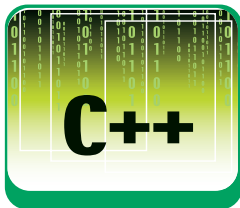
1) Al compilatore venga aggiunta la voce "C:\gc\include" fra le "additional include directories".
2) Al linker vengano aggiunte le voci "c:/gc6.7/gc.lib kernel32.lib user32.lib $(NoInherit)" fra le "additional library directories".

Se qualcosa è andato storto o siete molto pigri, potete provare a prendere il CD di ioProgrammo e aprire il file allegato. Vi ho preparato la libreria già compilata e un file di progetto "modello" già pronto: se siete fortunati, potrebbero funzionare.
Se entrambi gli approcci falliscono, potete provare a proporre il vostro problema sul forum di ioProgrammo; se, invece, avete successo, nulla vi vieta di scrivermi un'email piena di calorosi ringraziamenti!

TEST DEL GARBAGE COLLECTOR

Proviamo ad analizzare un piccolo programma, che serva da introduzione e test per il GC.

RIFERIMENTI

L'ultima versione del collector (attualmente la 6.7):
http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_source/gc.tar.gz

Visual Studio, Platform SDK e integrazione fra i due:
http://msdn.microsoft.com/vstudio/express/visualc/usingpsdk/

```
#define GC_NOT_DLL
#include <gc.h>
#include <gc_cpp.h>
int* pGlobal;
int main()
{
        GC_INIT();
        for (int i=0; 1000; i++) {
                //Test dell'interfaccia C
                int* n =
                        (int*)GC_MALLOC(sizeof(n));
                //Test dell'interfaccia C++
                int pGlobal = new (GC) int;
        }
        GC_gcollect();
}
```

Spiegherò le varie funzioni nei prossimi paragrafi, ma innanzitutto mi preme far notare alcuni appunti.

1) Definizione di GC_NOT_DLL: Si tratta di una delle tante variabili d'ambiente impostabili per condizionare il comportamento del GC. Nel caso specifico, con GC_NOT_DLL si dichiara che vogliamo usare una libreria statica, e non dinamica. Se avete compilato con il normale makefile (e state pertanto usando gc.lib) dovete richiamarlo prima di includere i file header, o sarete probabilmente sommersi dagli errori da parte del linker.

2) Inclusione degli header. Vi ritroverete ad usare principalmente questi due header: <gc.h> e <gc_cpp.h>. Il garbage collector espone, infatti, due interfacce: una serie di funzioni C, sostitutiva di malloc, realloc e free, e uno strato superiore C++, che sovraccarica il normale significato di new e delete, e definisce alcune classi da cui ereditare. Nei prossimi paragrafi vedremo una panoramica di entrambi gli approcci.

ALLOCAZIONE DI MEMORIA GESTITA

Cominciamo ad analizzare l'interfaccia C esposta dal BDW-GC. Le istruzioni fondamentali sono quelle che permettono di allocare memoria "gestita":

void* GC_MALLOC(size_t) È la principale funzione di allocazione nel GC, che vi ritroverete ad usare la maggior parte delle volte. La memoria sarà individuata dal garbage collector, che la richiamerà automaticamente quando non sarà più raggiungibile.

void* GC_MALLOC_ATOMIC(size_t): È equivalente a GC_MALLOC. Quando richiamate questa funzione di allocazione, però, promettete solennemente al garbage collector che il tipo di dati che state usando non contiene alcun puntatore significativo a memoria dinamica. Ciò permette al garbage collector di minimizzare i tempi di allocazione e di marcatura, evitandogli inutili ispezioni interne e l'inizializzazione del blocco.

void* GC_REALLOC(void*, size_t): È l'equivalente di realloc della libreria standard C, e permette di riallocare memoria in un certo blocco. Se è possibile il blocco si espanderà per esaudire la richiesta di nuovo spazio, altrimenti i suoi dati saranno ricopiati in un altro blocco più grande. Ovviamente, a differenza di realloc, il blocco sarà automaticamente gestito dal garbage collector.

Ecco un esempio di allocazione e riallocazione. È interessante studiarne l'output per vedere come la garbage collection avvenga periodicamente, e quindi i vecchi indirizzi vengano ciclicamente riutilizzati.

```
#include <gc.h>
#include <stdio.h>
#include <string.h>
int main()
{
        for (int i=0; i<1000; i++) {
                //una stringa e' un buon blocco da
                                allocare atomicamente
                char *stringa =
                  (char *)GC_MALLOC_ATOMIC(10);
                strcpy(stringa, "ciao!\n");

                //stampiamo l'indirizzo e il
                                contenuto della stringa
                printf("(%d) %s", stringa, stringa);

                //rendiamo la stringa più capiente
                stringa =
                  (char *)GC_REALLOC(stringa, 20);
                strcpy(stringa, "ciao, mondo!\n");

                //stampiamo l'indirizzo e il
                            contenuto della nuova stringa
                printf("(%d) %s", stringa, stringa);
        }
}
```

ALLOCAZIONE E DEALLOCAZIONE DI MEMORIA "NON GESTITA"

Notate che usando il GC non c'è ragione di invocare le funzioni di libreria malloc, realloc e free. E, anzi, è vietato farlo, dal momento che si potrebbe andare a scombinare l'area di memoria del garbage collector,

con effetti disastrosi. Se volete gestire alcune allocazioni (o deallocazioni) da voi, il GC vi fornisce le seguenti funzioni "sicure":

void* GC_MALLOC_UNCOLLECTABLE(size_t): Questa funzione permette di allocare memoria individuabile dal garbage collector, tuttavia questa non sarà richiamata, se non alla fine dell'applicazione. Equivale, in sostanza, ad avere un puntatore globale perennemente indirizzato alla risorsa.

void GC_FREE(void*): Se proprio insistete nel voler deallocare una risorsa incuranti del parere del garbage collector, potete usare questa funzione. Normalmente non avrete alcun bisogno di farlo, dal momento che non c'è fondamentalmente alcun vantaggio in termini di presazioni (anzi, per piccoli oggetti è sicuramente meglio lasciar fare all'allocatore del GC). Se avete, però, oggetti molto grandi che morite dalla voglia di distruggere in modo deterministico, questa funzione fa per voi.

L'esempio che segue mostra queste due funzioni in azione, evidenziando come vadano spesso in coppia. Provate a scrivere questa versione, e la sua corrispondente con *GC_MALLOC* e senza *GC_FREE*. Che cambiamento notate negli indirizzi?

```
#include <gc.h>
#include <stdio.h>
#include <string.h>
int main()
{
        for (int i=0; i<1000; i++) {
                //allochiamo una stringa
                                        "non gestita"
                char *stringa =
            (char *)GC_MALLOC_UNCOLLECTABLE(10);
                strcpy(stringa, "ciao!\n");

                //stampiamo l'indirizzo e il
                                contenuto della stringa
                printf("(%d) %s", stringa, stringa);

                //forziamo la deallocazione della
                                                stringa
                GC_FREE(stringa);
        }
}
```

FUNZIONI ACCESSORIE E DI DEBUG

Il garbage collector espone anche diverse funzioni per l'ottimizzazione e il debug. Non potendo vederle tutte, ci limiteremo alle due seguenti.

void GC_gcollect(): "Consiglia" al GC di eseguire una garbage collection. Normalmente il GC accetta il suggerimento, ma non c'è da contarci in modo assoluto. La maggior parte delle volte, non avrete alcun bisogno di richiamare una collection esplicitamente: ci penserà il GC stesso ad attivarsi se e quando sarà il caso.

size_t GC_get-_heap_size(): Restituisce le dimensioni correnti dello heap associato al garbage collector.

Potete provare a ritoccare gli esempi precedenti, stampando a video l'heap size, e/o forzando delle raccolte con gcollect. Questo genere di debug è utile per capire effettivamente cosa succede dietro le scene.

OPERATORI ::NEW E ::DELETE

John R. Ellis e Jesse Hull hanno lavorato ad un'interfaccia che permetta di integrare il BDW-GC nel C++. Non è stata un'impresa facile, dal momento che in C++ esistono così tanti modi di allocare le risorse, che i vari compilatori faticano nel seguir bene lo standard. Se avete comprato l'IoProgrammo del mese scorso (bravi!) vi consiglio di riprendere in mano la vecchia puntata, perché può tornarvi utile.
Innanzitutto, il file <gc_cpp.h> si occupa di fornire una nuova implementazione per gli operatori ::new e ::delete, nascondendo quella originale, in modo che scrivere:

```
int *p = new int;
delete p;
```

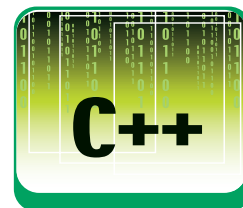
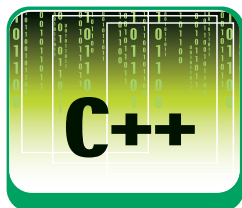
Sia sostanzialmente uguale a scrivere:

```
int *p = GC_MALLOC_UNCOLLECTABLE(sizeof(int));
GC_FREE(p)
```

Il "sostanzialmente" è d'obbligo, perché la versione C non si occupa di costruire il dato (vi ricordate la discussione sul Placement New per antonomasia della scorsa puntata?).L'operatore new globale, quindi, alloca memoria raggiungibile ma che **non verrà raccolta automaticamente dal garbage collector**. Questo vuol dire che lo userete poco!
Se volete allocare invece della memoria gestita, dovete usare un placement new che preveda l'enumerazione "GC" come argomento:

```
int *p = new (GC) int;
```

che è "sostanzialmente" uguale a scrivere:

```
int *p = GC_MALLOC (sizeof(int));
```

OPERATORI GC::NEW E GC::DELETE

Un altro modo (che non abbiamo ancora visto, in questa serie) per sovraccaricare gli operatori new e delete, è farlo all'interno di una data classe T. Così, tutti gli oggetti di tipo T (o derivati da T), useranno gli operatori new e delete esposti dalla classe. Il BDW-GC ne approfitta per fornire una classe chiamata ***gc***, derivando dalla quale è possibile utilizzare il semplice "new" per allocare della memoria gestita. Definendo, ad esempio, un tipo come:

```
struct Persona : gc
{ /* definizione… */};
```

queste due istruzioni saranno equivalenti:

```
Persona *p = new Persona;
Persona *p = ::new(GC) Persona;
```

Se, invece, si è ereditato dalla classe gc, ma in un particolare contesto si desidera allocare memoria non richiamabile, è possibile usare il placement new con parametro NoGC. Queste due istruzioni sono, quindi, equivalenti:

```
Persona *p = new(NoGC) Persona;
Persona *p = ::new Persona;
```

ROBERTO ALLEGRA

Il sito www.robertoallegra.it contiene l'elenco degli articoli pubblicati in questa rubrica, con eventuali approfondimenti e errata corrige, e la possibilità di contattare l'autore per richieste, critiche e suggerimenti.

GC_CLEANUP E FINALIZZATORI

Finora ci siamo dedicati sostanzialmente all'allocazione. Ma che dire della deallocazione e della distruzione delle risorse? Nella programmazione tradizionale siamo abituati a scrivere istruzioni che rilascino le risorse che abbiamo acquisito. Il C++ ha introdotto i distruttori proprio per quest'unico scopo.
Poiché il GC dealloca da sé la memoria, prevederne il rilascio non è più necessario. Inoltre la deallocazione delle risorse avviene in maniera non-deterministica, quando il GC "si accorge" che queste non servono più – cosa che può avvenire in qualunque momento, spesso molto dopo rispetto al rilascio dell'ultimo puntatore. Per tutti questi motivi, il garbage collector non richiama alcun distruttore sulle risorse che dealloca. Tuttavia il concetto di "risorse" non si limita semplicemente alla "memoria", ma si estende ai lock, ai file, all'accesso alle periferiche, e molto oltre. Il garbage collector non sa come rilasciare questo genere di risorse! Per questo, è possibile indicare, all'allocazione dell'oggetto, un **finalizzatore**, ovverosia una routine che sarà chiamata dal garbage collector prima della deallocazione dell'oggetto. Il finalizzatore deve avere la marcatura "void nome(void*, void*)", laddove il primo argomento è l'oggetto deallocato, e il secondo è un dato opzionale che è possibile indicare al momento dell'allocazione.

```
//… #includes
struct Persona {/* definizione */};
void finalizzatore(void* obj, void* data)
{
        Persona* persona =
                reinterpret_cast<Persona*>(obj);
        cout << persona->nome_ << " e' morto" <<
                (char*)data << endl;
}
int main()
{
        for (int i=0; i<100; i++) {
                new(GC, finalizzatore, " per la prima
                volta.") Persona("Priamo");
                new(GC, finalizzatore, " di nuovo!")
                Persona("Priamo");
        }
        GC_gcollect();
}
```

Osservate bene l'output dell'applicazione: le due frasi "per la prima volta" e "di nuovo" non si intervallano come ci si aspetterebbe, ma appaiono alla rinfusa. È normale: il GC non dà alcuna garanzia sul come e sul quando un dato sarà deallocato.
Per rendere le cose molto più semplici e circoscritte, il BDW-GC fornisce la classe gc_cleanup, che permette di specificare un distruttore da usare come finalizzatore:

```
struct Persona : gc_cleanup {
        string nome_;
        Persona(string nome) : nome(nome_) {};
        ~Persona() { cout << nome_ << " è
                morto!\n";}
};
int main() {
        for (int i=0; i<100; i++) {
                new Persona("Priamo");
        }
        GC_gcollect();
}
```

CONCLUSIONI

Con questa puntata si conclude la nostra serie sul memory management in C++. Si è trattato indubbiamente di un giro panoramico, peraltro ristretto: small-allocators per piccoli oggetti, arene, accesso alle funzionalità esposte da un sistema operativo… sono tutti regni che rimarranno, almeno per ora, inesplorati.

Roberto Allegra

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

C#

SOFTWARE RESIDENTE

Come posso sapere se un'applicazione che gira in Background va in Crash?

```
using System.Diagnostics;
public void WriteEventLog(string sCallerName, string sLogLine)
{
  if (!System.Diagnostics.EventLog.SourceExists(sCallerName))
    System.Diagnostics.EventLog.CreateEventSource(sCallerName,
"Application");
  EventLog EventLog1 = new EventLog();
  EventLog1.Source = sCallerName;
  EventLog1.WriteEntry (sLogLine, EventLogEntryType.Warning);
}
```

CONTRLLO SULLE ISTANZE

Come posso evitare che vengano eseguite più istanze della stessa applicazione?

```
using System.Threading;
static void Main()
{
  bool bAppFirstInstance;
  oMutex = new Mutex(true, "Global\\" + â€œYOUR_APP_NAMEâ€_,
                                   out bAppFirstInstance);
  if(bAppFirstInstance)
    Application.Run(new formYOURAPP() or classYOURAPP());
  else
    MessageBox.Show("The threatening message you want to go
                                                  forâ€_",
        "Startup warning",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation,
        MessageBoxDefaultButton.Button1);
}
```

POSTA ELETTRONICA

Come posso inviare una email da C# ?

```
using System.Web.Mail;
private bool SendEmail(string sFrom, string sTo, string sCC,
    string sBCC, string sSubject, string sMessage, int iMailType)
{
  try
  {
    MailMessage Message = new MailMessage();
    // If sFrom is blank, system mail id is assumed as the sender.
    if(sFrom=="")
      Message.From = "default@myserver.com";
    else
      Message.From = sFrom;
    // If sTo is blank, return false
    if(sTo=="")
      return false;
    else
      Message.To = sTo;

    Message.Cc = sCC;
    Message.Bcc = sBCC;
    Message.Subject = sSubject;
    Message.Body = sMessage;
    Message.BodyFormat = MailFormat.Text;
    SmtpMail.SmtpServer = "Put a valid smtp server IP";
    SmtpMail.Send(Message);

    return true;
  }
  catch(System.Web.HttpException ehttp)
  {
    // Your exception handling code here...
    return false;
  }
  catch(Exception e)
  {
    // Your exception handling code here...
    return false;
  }
  catch
  {
    // Your exception handling code here...
    return false;
  }
}
```

INFORMAZIONI SULLA RETE

Come posso ottenere l'IP dall'host name?

```
using System.Net;
public string GetIPAddress(string sHostName)
{
  IPHostEntry ipEntry = Dns.GetHostByName(sHostName);
  IPAddress [] addr = ipEntry.AddressList;
  string sIPAddress = addr[0].ToString();
  return sIPAddress;
}
```

TENERE TRACCIA DEGLI ERRORI

Come posso scrivere informazioni in un file di log?

```
using System.IO;
public void WriteLogLine(string sCallerName, string sLogFolder,
                        long lCallerInstance, string sLogLine)
{
  lock(this)
  {
    string sFileName;
    sFileName = String.Format("{0}_{1:yyyy.MM.dd}_{2:00}.log",
                  sCallerName, DateTime.Now, lCallerInstance);
    StreamWriter swServerLog =
          new StreamWriter(sLogFolder + sFileName, true);
    swServerLog.WriteLine(
          String.Format("[{0:T}] {1}", DateTime.Now, sLogLine));
    swServerLog.Close();
  }
}
```

INTERFACCIA GRAFICA

Come posso muovere una finestra senza bordo?

```
Private diff As Point

Public Sub New()
InitializeComponent
End Sub

Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
MouseEventArgs)
diff = New Point(-e.X, -e.Y)
End Sub

Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
MouseEventArgs)
If e.Button = MouseButtons.Left Then
Dim mouse_loc As Point = Control.MousePosition
mouse_loc.Offset(diff.X, diff.Y)
Me.Location = mouse_loc
End If
End Sub
```

GESTIONE DEL TESTO

Come posso calcolare il numero di occorrenze in una stringa?

```
Public Function numOccorrenze(ByVal myStr As String, ByVal
      myWord As String, Optional ByVal isCaseSensitive As Boolean =
                                              False) As Integer
Dim myArray() As String = myStr.Split
Dim mySimbol As String = ",';.: !?"

numOccorrenze = 0
For x As Integer = 0 To UBound(myArray)
'Pulisco l'array dai segni di punteggiatura
For y As Integer = 0 To mySimbol.Length - 1
myArray(x) = myArray(x).Replace(mySimbol.Substring(y, 1), "")
Next
'Confronto la stringa
If isCaseSensitive Then
If myArray(x).CompareTo(myWord) = 0 Then numOccorrenze += 1
Else
If myArray(x).ToUpper.CompareTo(myWord.ToUpper) = 0 Then
                                          numOccorrenze += 1
End If
Next
Return numOccorrenze
End Function
```

FEEDBACK ALL'UTENTE

Come creare MessageBox personalizzate?

```
Dim f2 As Form = New Form2
f2.ShowDialog()
If f2.DialogResult = DialogResult.OK Then
MessageBox.Show("Hai premuto Sei bello")
Else
MessageBox.Show("Hai premuto Sei Brutto")
```

PERSISTENZA DEI DATI

Come posso prelevare un valore dal registry?

Si, l'esempio d'uso è il seguente:

```
Imports Microsoft.Win32
Function GetDBConnect() As String
Dim pobjRegKey As RegistryKey
Dim pstrValue As String
 pobjRegKey = _

Registry.LocalMachine.OpenSubKey("SYSTEM\CurrentControlSet\Serv
                               ices\MyService\Parameters", False)
 If Not IsNothing(pobjRegKey) Then
```

```
    pstrValue = pobjRegKey.GetValue("DBConnect",
                                vbNullString).ToString
    Return pstrValue
    Else
      Return vbNullString
  End If
End Function
?>
```

PERCORSI LUNGHI E CORTI

Come posso convertire un path lungo nella sua versione "short"

```
Imports System.Text
Imports System.Runtime.InteropServices

  <DllImport("kernel32.dll", SetLastError:=True,
CharSet:=CharSet.Auto)> _
  Public Shared Function GetShortPathName(ByVal lpszLongPath As
                                                String, _
            ByVal lpszShortPath As StringBuilder, ByVal cchBuffer As
                                        Integer) As Integer
  End Function

  Private Function ConvertLongPathToShort(ByVal longPathName As
                                        String) As String
    Dim shortNameBuffer = New StringBuilder

    Dim size As Integer = Me.GetShortPathName(longPathName,
                                        shortNameBuffer, _

shortNameBuffer.Capacity)
    If (size >= shortNameBuffer.Capacity) Then
      shortNameBuffer.Capacity = size + 1
      GetShortPathName(longPathName, shortNameBuffer,
                                shortNameBuffer.Capacity)
    End If

    Return shortNameBuffer.ToString()
  End Function
```

JAVA

INTERAGIRE CON IL SISTEMA

Come posso catturare uno ScreenShot dello schermo?

```
import java.awt.AWTException;
import java.awt.Rectangle;
import java.awt.Robot;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
```

```
public class RobotExp {

    public static void main(String[] args) {

        try {

            Robot robot = new Robot();
            // Capture the screen shot of the area of the screen defi-
ned by the rectangle
            BufferedImage bi=robot.createScreenCapture(new
Rectangle(100,100));
            ImageIO.write(bi, "jpg", new File("C:/imageTest.jpg"));

        } catch (AWTException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MATEMATICA OTTIMIZZATA

Come posso trovare il massimo comun divisore con un algoritmo ricorsivo?

```
 public static long gcd(long a, long b) {
   if (b==0)
     return a;
   else
     return gcd(b, a % b);
 }
```

TRASMETTERE LE INFORMAZIONI

Come posso passare un argomento da linea di comando ad un programma java?

```
public class CmdLnArgmntExp {
    public static void main(String[] args) {
        System.out.println("d");
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

PHP

LAVORARE CON APACHE

Come posso sapere quali moduli di apache sono installati?

```
<?php
print_r(apache_get_modules());
?>
```

ECLIPSE + BORLAND = JBUILDER 2007

BORLAND HA DECISO DI ADOTTARE ECLIPSE COME BASE PER LO SVILUPPO DEI NUOVI IDE PER JAVA. ATTORNO CI HA COSTRUITO UN'INFRASTRUTTURA DI TUTTO RISPETTO AGGIUNGENDO DECINE DI PLUGIN. ECCO I DETTAGLI...

Borland, da sempre sinonimo di strumenti di sviluppo per programmatori, ultimamente ha dirottato i suoi interessi verso soluzioni middleware e orientate allo sviluppo in team (fornendo strumenti per la modellazione, generazione del codice, integrazione di processi e così via). Però non ha del tutto abbandonato il mercato degli IDE: ha infatti creato un'azienda da lei controllata, chiamata CodeGear, a cui lascia l'eredità dello sviluppo di IDE. L'ultimo nato è JBuilder 2007, il rinnovato IDE per lo sviluppo di applicazioni Java. A differenza delle precedenti versioni, la piattaforma su cui si basa non è più sviluppata in proprio, ma si appoggia ad Eclipse. Eclipse non ha certo bisogno di molte presentazioni: è unanimamente riconosciuto come un'ottima piattaforma Open Source che ha fatto della flessibilità e dell'estendibilità (grazie ad un'architettura a plug-in) il proprio cavallo di battaglia. Ecco che JBuilder 2007 è un "super Eclipse", in cui, sulla base di Eclipse 3.2 versione standard, integra i più diffusi plug-in open source e ne affianca di propri dedicati allo sviluppo in team.

LE VERSIONI DI JBUILDER

JBuilder si presenta in tre versioni: Developer, Professional ed Enterprise. In questo contesto si farà riferimento alla versione Enterprise, quella più completa e ricca di funzionalità.

INSTALLAZIONE ED ATTIVAZIONE

JBuilder richiede un computer piuttosto potente per poter funzionare in maniera soddisfacente. Basti pensare che è richiesto almeno 1 GB di RAM per le versioni Professional ed Enterprise. InterBase 2007 e JDataStore 7. Attenzione al fatto che InterBase viene rilasciato, in questo contesto, con licenza "developer": questo significa che lo si può usare nello sviluppo di applicazioni ma non lo si può installare su un server remoto, presso un cliente (quest'ultimo dovrà acquistare una licenza a parte).

Durante l'installazione viene chiesto anche se si vuole installare uno tra i seguenti application server: Apache Geronimo, JBoss o GlassFish; oppure è possibile installare il servlet container

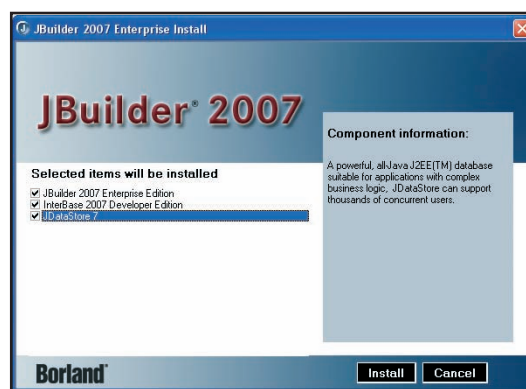



Figura1: InterBase e JDataStore

Apache Tomcat.

Eventuali server installati verranno configurati affinché vengano integrati in maniera semplice con JBuilder (a dire il vero un minimo di configurazione è comunque necessaria. Infatti basterà, dopo l'installazione, creare un progetto

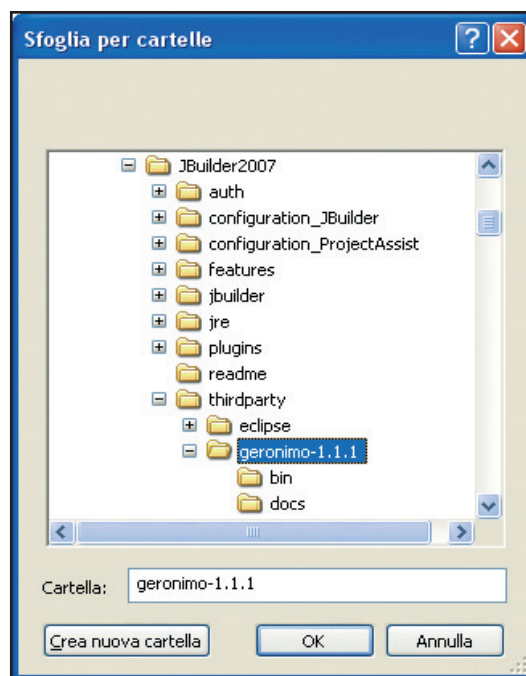



Figura 2: cartella ove vengono installati eventuali Application Server.

EJB per accorgersi che eventuali Application Server installati non vengono automaticamente riconosciuti dall'ambiente; per referenziarli, far riferimento alla cartella "thirdparty" sotto la cartella di installazione del prodotto).
Una volta installato il prodotto, lo si può attivare fornendo una chiave appropriata. Essa permetterà di registrare il prodotto e, per concludere l'attivazione, è necessario collegarsi ad internet o richiedere un file di attivazione da scaricare da un indirizzo di email.
Alla prima esecuzione è possibile specificare il workspace di riferimento (esso conterrà tutti i progetti creati). È possibile far sì che la scelta sia considerata il default per le prossime esecuzioni o che venga fatta di volta in volta.

QUALI PLUGIN?

Mandato in esecuzione l'editor si può iniziare con la verifica dei diversi plug-in installati. Per farlo si scelga: Help > About JBuilder. Dalla finestra si prema sul pulsante "Feature details".

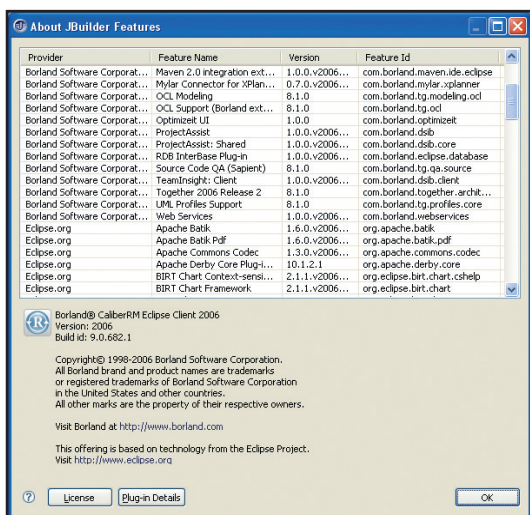



Figura 3: i diversi plug-in installati.

Nella colonna "Provider" vengono evidenziati i nomi delle aziende fornitrici. Come si può notare molte caratteristiche sono proprietarie di Borland, ma molte di più sono fornite da aziende di terze parti e, in particolare, provengono dal progetto Eclipse.

CREARE UN NUOVO PROGETTO

Provare ora a creare un nuovo progetto. Per farlo scegliere "**File > New > Project**". Si aprirà un wizard da cui è possibile scegliere il tipo di progetto da creare. I diversi tipi di progetto sono organizzati in una semplice struttura ad albero, la cui radice rappresenta la tipologia di riferimento. Supponendo di voler realizzare un Web Service, scegliere "**Web Services > Axis Web Service Project**". Premendo sul pulsante "**Next**" si passerà alla configurazione di base del progetto. La prossima schermata prevede il nome del progetto e la specifica un un ambiente di runtime appropriato. Inizialmente non c'è alcun ambiente configurato (piccola nota dolente: visto che l'installazione di JBuilder prevedeva degli ambienti di default, il minimo che ci si poteva aspettare era la loro presenza in questo contesto!). Configurato l'application server di riferimento (o altro ambiente) si possono lasciare tutte le altre opzioni di default e si può premere sul tasto "Finish". Il nuovo progetto viene creato e viene chiesto se si desidera aprire la "modeling perspective".

TANTE VISTE PER GLI OGGETTI

Quando viene creato un determinato progetto, viene chiesto se si vuole adottare una particolare "perspective", quella associata di default al progetto creato (per un progetto Web Service, si è visto che viene usata la "modeling perspective"). Tali "perspective" altro non sono che particolari viste sugli oggetti esistenti. Una vista contiene un insieme predefinito di visualizzatori (alcuni gerarchici altri grafici altri plain-text), con una loro posizione sullo schermo. Ogni perspective può essere a sua volta personalizzata, sia con gestendo ulteriori visualizzatori che riposizionandoli a piacere sullo schermo. Per cambiare perspective o, più semplicemente, per sapere quali perspective si ha a disposizione, scegliere "**Window > Open Perspective > Other**".

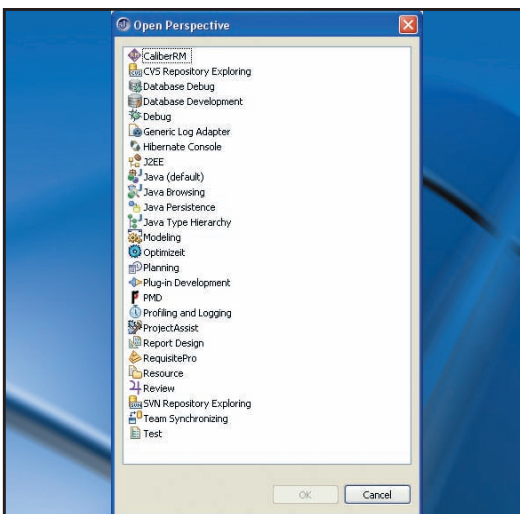
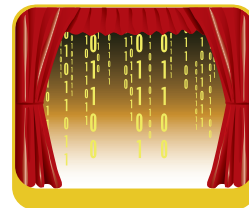
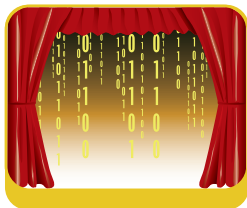



Figura 4: le perspective disponibili in JBuilder 2007.

IMPORT DI PROGETTI ESISTENTI

JBuilder permette sia di importare progetti esistenti creati dalle versioni precedenti di JBuilder, sia altri progetti creati con Eclipse. Nel primo caso si deve utilizzare una delle procedure guidate messe a disposizione dall'IDE (procedure realizzare ad hoc da Borland), nel secondo caso è sufficiente importare il progetto nel proprio workspace (infatti, essendo JBuilder basato su Eclipse, non ci sono particolari conversioni né adattamenti da fare).

Chi ha dimestichezza con Eclipse standard potrà notare numerose perspective nuove e peculiari di JBuilder 2007 (alcune in realtà sono proprie di plug-in di terze parti, ma non installati in Eclipse di default; è il caso, per esempio, di "Hibernate Console", proveniente da un plug-in fornito da JBoss).

I MODELER VISUALI

Se si è risposto affermativamente all'uso della modeling perspective, ci si trova in un ambiente in cui è presente un modellatore visuale su cui intervenire per la creazione del Web Service.
Accanto al modeler visuale si può notare, sulla sinistra, un insieme di oggetti creati in automatico dal Wizard (in questo caso si possono notare vari oggetti che corrispondono ad una WebApplication che fa uso di Axis).
La perspective di modellazione visuale è una delle novità di JBuilder: infatti esso fornisce la possibilità di creazione (e modifica) sia di Web Services che di EJB in modalità visuale. A partire da tale modello, vengono generate le classi Java corrispondenti. Non solo: eventuali modifiche alle classi vengono sincronizzate con il modello di riferimento e anch'esso resta aggiornato. Tale caratteristcia si apprezza anche su un altro plug-in di enorme valore:

Figura 5: modeler perspective per i Web Services.

Together. Esso è un modeler UML con le stesse proprietà di sincronizzazione degli altri modeler visuali.

NOTA

RIFERIMENTI

Esistono numerosi newsgroup gestiti direttamente da Borland. Per quelli specifici di JBuilder far riferimento alla gerarchia borland.public.jbuilder .*

CREARE WEB SERVICES?

Chi sviluppa Web Services sa che si possono creare o partendo da un WSDL (modalità usata sempre nel caso di client verso Web Services esistenti, ma consigliata anche dai "puristi" della programmazione Web Services) oppure da classi Java che verranno esposte come Web Services. Vediamo come i due casi possono essere usati in JBuilder.
Supponendo di avere a disposizione un WSDL, si può avere la necessità di creare un client, nel caso esso si riferisca ad un Web Service esistente, o le classi server, se esso è solo il "contratto" definito a priori in fase di analisi o dopo opportuni accordi con il cliente. In ogni modo, partendo dal modeler visuale, fare clic sul pulsante "**WSDL Web Service**". Il cursore cambia forma e, facendo clic sulla griglia di design, si può inserire un nuovo oggetto. Esso va personalizzato con un nome appropriato. Sulla finestra in basso appaiono le "**properties**" dell'oggetto: agendo su tali valori è possibile personalizzarle. In particolare sul tab "Input", c'è la proprietà "Input WSDL file". Basterà fare clic sul pulsante di selezione per aprire una finestra di dialogo da cui scegliere il file WSDL di partenza. Effettuata la scelta, in automatico, il modeler genera le classi (visibili sulla gerarchia a sinistra) e modifica la visualizzazione grafica associata. Intervenendo sulle

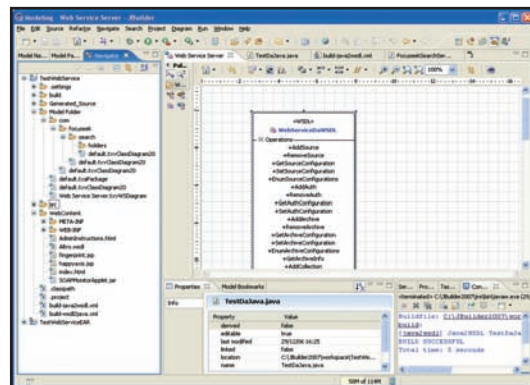



Figura 6: Creazione di un Web Services a partire dal WSDL.

altre properties è possibile personalizzare ciascun aspetto del Web Service (in particolare con un semplice flag è possibile decidere se deve essere generata la parte client o server del servizio). È interessante osservare che, in automatico, vengono anche generati dei test attraverso JUnit.

PARTIRE DA UNA CLASSE JAVA

Chi sviluppa Web Services sa quanto può essere complesso partire dal WSDL. Molto più spesso conviene partire da una classe Java (o da un'interfaccia) e far generare dalle signature dei metodi il WSDL corrispondente. Pertanto, dalla gerarchia a sinistra, sotto "src", scegliere "New > Other > Class". Dato il nome alla classe, concludere il wizard. Dichiarare,

per esempio, due attributi privati: uno di tipo String (chiamato descrizione), l'altro di tipo int (chiamato quantità), quindi generare i getter/setter per ciascuna proprietà (operazione automatizzabile facendo clic con il pulsante destro del mouse e scegliendo "Source > Generate Setters and Getters ...").

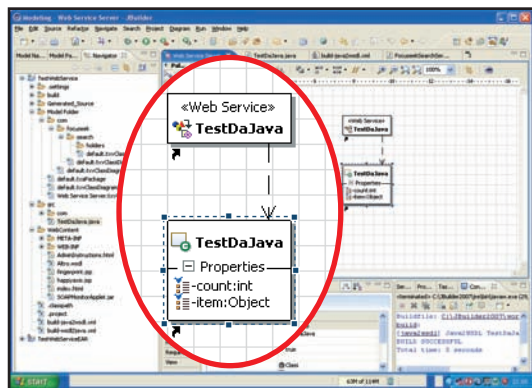
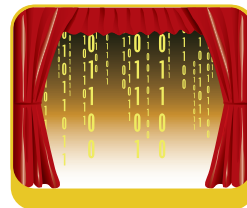



Figura 7: Creazione di un Web Services a partire da una classe Java.

SOTTO SOTTO C'È AXIS

Le operazioni di base sono tutte realizzate grazie al framework Axis (infatti esso mette a disposizione tutte le utility sia per la creazione di stub a partire dal WSDL, sia per la creazione del WSDL a partire da classi Java); ma, come si può osservare, lo sviluppo di nuovi Web Services è alquanto semplificato dall'interfaccia intuitiva e davvero semplice messa a disposizione da JBuilder. Esiste inoltre una terza modalità di creazione di Web Services: essa si basa su EJB esistenti esposti come Web Services. Anche in questo caso le operazioni per portare a termine lo sviluppo sono analoghe ai casi presentati. La modalità di modellazione visuale viene usata anche per lo sviluppo degli EJB secondo la versione 3 dello standard. JBuilder permette anche di migrare in maniera semplice, grazie a procedure guidate, da EJB 2 ad EJB 3.

LA COLLABORAZIONE AL CENTRO!

Tra i numerosi plugin presenti in JBuilder buona parte si rivolgono alla gestione di progetti sviluppati da team di sviluppo. Accanto a repository per la sincronizzazione del lavoro (CVS e subversion) c'è la possibilità di gestire configurazioni di progetti condivise, creare task list e assegnarle a diversi sviluppatori (comprese eventuali gestioni di bug, features e desiderata) nonché possibilità di monitorare l'avanzamento del progetto (sia con grafici che con valori di sintesi quali metrica sul software, build continuativo e test in corso d'opera). Il tutto ben si adatta ai più moderni dettami della Extreme Programming ma, ovviamente, possono essere usati con profitto anche in altri tipi di processi di gestione del ciclo di vita di un'applicazione.

ALTRE INFORMAZIONI

Nella cartella di installazione di JBuilder sono presenti due documenti PDF. Uno è un quick start, l'altro un tutorial di oltre 300 pagine. A dire il vero quest'ultimo denota poca cura: alcune parti sono duplicate (c'è in doppio "Getting started" iniziale del tutto simile), altre solo abbozzate.
Sul sito CodeGear sono presenti altri documenti che riassumono sia le caratteristiche salienti, sia risposte alle domande più frequenti. Chi è interessato a una panoramica sulle caratteristiche del prodotto (e conosce bene l'inglese!) può guardare i video presenti su YouTube. Basta collegarsi alla pagina http://www.youtube.com e ricercare "jbuilder 2007"; da lì vengono reperiti motli video, di cui alcuni molto interessanti (sono registrazioni di conferenze tenute da esperti di CodeGear).

CONCLUSIONI

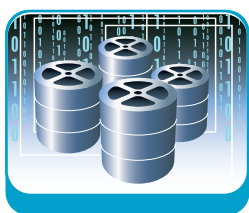
JBuilder rappresenta senz'altro un ottimo strumento, utile per minimizzare i tempi di startup di nuovi progetti, di cui consente un monitoraggio puntuale ed efficace, massimizzando la collaborazione tra team, anche remoti. Questo è indispensabile per tutte le realtà complesse e articolate. Questi aspetti sono invece molto meno influenti nel caso di team di sviluppo ridotti e collocati all'interno della stessa unità organizzativa. In questi casi, probabilmente, il costo elevato della licenza (poco meno di 2000 dollari per la versione enterprise, 800 per quella professional e 400 per quella developer) di un prodotto così ricco e completo può essere motivo di forti riserve, preferendo ambienti più leggeri ed economici. In ogni modo si spera che la scelta dell'adozione di Eclipse come base per il nuovo IDE possa portare ad un vantaggio di tutte le parti in causa: ulteriori sviluppi di Eclipse stesso da un lato (grazie al contributo di Borland), minor tempo per partire ed essere produttivi con JBuilder qualora si decidesse di passare da Eclipse a tale ambiente. Infine un suggerimento per chi usa Eclipse e, pur valutando JBuilder, non lo ritenga adatto ai propri scopi: la presenza di tanti plug-in di qualità può aiutare nella scelta della configurazione del proprio ambiente di sviluppo guardando ad essi e prendendone spunto.

Ivan Venuti

ECLIPSE SUL WEB

Per informazioni specifiche su Eclipse far riferimento al sito web ufficiale: www.eclipse.org

OTTIMIZZAZIONE DEI REPORT

DOPO AVER INTRODOTTO UN'ARCHITETTURA PER ESEGUIRE I REPORT LATO SERVER. IMPLEMENTIAMO UNA SOLUZIONE CHE CI PERMETTA DI CONTROLLARE E BLOCCARE L'ELABORAZIONE DEL REPORT PROPRIO COME SE GIRASSE LATO CLIENT.

Nel numero precedente di ioProgrammo abbiamo introdotto una semplice architettura che ci consente di elaborare un report tramite un modulo della nostra applicazione risiedente sul server. Il report così elaborato può essere inviato al client attraverso la tecnologia .NET Remoting in modo che possa essere visualizzato e stampato dall'utente. Questa tecnica sgrava il client dal pesante onere dell'elaborazione e dell'accesso a risorse pregiate come il database.

Nella maggior parte dei casi questa soluzione è completa e sufficiente. Tuttavia questo approccio introduce una limitazione, sia per lo sviluppatore sia per l'utente, rispetto al modulo "tutto in locale". Viene cioè ridotta la possibilità di controllo e di interazione tra l'applicazione client e il report durante le fasi di elaborazione di quest'ultimo. Apparentemente potrebbe apparire una finta limitazione, ma basta un semplice esempio per capire che purtroppo non è così. L'elaborazione di un report può durare a volte anche diversi minuti, persino decine di minuti, se devono essere elaborate migliaia di pagine, magari di un report molto complesso perché dotato di numerosi subreport o comunque di logica "pesante". Cosa accade se l'utente, dopo aver lanciato un report da mezz'ora di elaborazione, si accorge di essersi dimenticato un piccolo parametro da passare? In un'architettura tradizionale con il report "in pancia" nel client, probabilmente sarebbe sufficiente chiudere la maschera da cui l'utente ha fatto partire la richiesta. Oppure, nella peggiore delle ipotesi, l'utente potrebbe chiudere l'intera applicazione client e il report in esecuzione si arresterebbe di conseguenza. Infine, se proprio dovesse risultare duro a morire, esiste sempre il task manager col suo ammazzaprocessi...

Ma cosa accade se il report in questione viene elaborato dal server che restituisce il controllo (e il risultato) al client solo al termine dell'elaborazione? Succede che l'utente non ha modo di bloccarne l'elaborazione e, anche qualora dovesse riuscire a terminare l'applicazione client, il report continuerebbe comunque a girare sul server fino al termine della sua elaborazione, secondo i tipi canoni operativi di .NET Remoting. Certo, si può terminare il Report Server, ma innanzitutto non è molto "civile" pensare che un utente debba riavviare il modulo server della propria applicazione e, in seconda battuta, si deve considerare la possibilità che il server stia contemporaneamente rispondendo a richieste provenienti da altri client. Come risolvere, dunque, il problema? Trovare una soluzione sarà lo scopo del presente articolo...

L'ARCHITETTURA DI REPORTISTICA

Lo sviluppo della nostra soluzione è realizzato in C# e dunque possiamo sviluppare l'intera soluzione con Microsoft Visual C# Express 2005 che è completamente gratuito.

Il prodotto di reportistica utilizzato è Datadynamics ActiveReports for .NET 3.0, una delle migliori soluzioni di reportistica per .NET grazie al suo ottimo rapporto qualità prezzo (la versione Standard, utilizzata in questi esempi, costa alcune centinaia di euro.

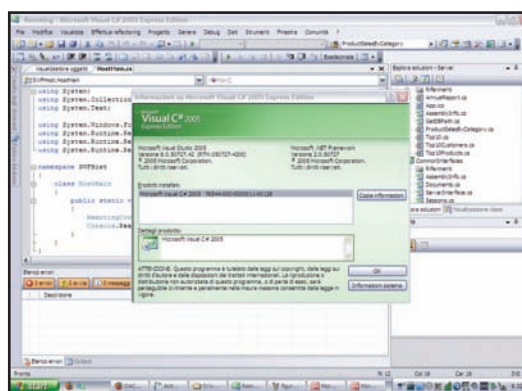



Fig. 1: Visual C# 2005 Express in tutto il suo splendore

Il cuore del sistema di reportistica lato server, introdotta nel numero precedente, prevede la defi-

nizione di un'interfaccia di riferimento per i Report Server:

```
namespace ioProgrammo.RemotingServices
{

        public interface IRemoteServer
        {
    //LOGIN
    Guid Login(string username, string password);

    //INVOCA REPORT REMOTO
    Stream ProcessReport(Guid sessionId, IDictionary
                                        parameters);
    Stream ProcessReport(Guid sessionId, IDictionary
                              parameters, Guid syncId);

    //SYNCK MANAGER
    bool SetSync(Guid sessionId, SyncInfo sync);
    SyncInfo GetSync(Guid sessionId, Guid syncId);
    bool RemoveSync(Guid sessionId, Guid syncId);
        }
}

namespace ioProgrammo.RemotingServices
{
  public class Session
  {
    public string Username = null;
    public Guid SessionId = Guid.Empty;
    public DateTime StartTime = DateTime.MinValue;
    public DateTime LastAccessTime =
                                   DateTime.MinValue;
  }
}
```

Il metodo fondamentale è ProcessReport che accetta in ingresso un dizionario di parametri da passare al report da mandare in esecuzione sul server e restituisce uno stream che contiene la serializzazione del report perfettamente elaborato e pronta per essere restituito al client chiamante che ne deserializza il contenuto e lo mostra in anteprima di stampa per l'utente. Il client deve avviare una comunicazione con il server invocandone il metodo login che permette a questi di identificarlo e di attribuirgli un Guid univoco che ne identifica la sessione. Il server conserverà questa informazione in un oggetto Session riposto in una collezione di Session che conserva tutte le sessioni correntemente aperte dai client per cui fa da servente.
Rivediamone, infatti, l'implementazione del metodo Login definito nella classe InterfaceServer che implementa proprio l'interface IRemoteServer:

```
private Dictionary<string, Session> _sessions = new
                          Dictionary<string, Session>();

```

```
public Guid Login(string username, string password)
{
  Hashtable logins = new Hashtable();
  logins["ciro"] = "wow";
  logins["ciro1"] = "wow";
  logins["pinuccio"] = "wow3";

  //controllare username e password
  if (logins.ContainsKey(username.ToLower()) &&
     logins[username.ToLower()].ToString() == password)
  {
```

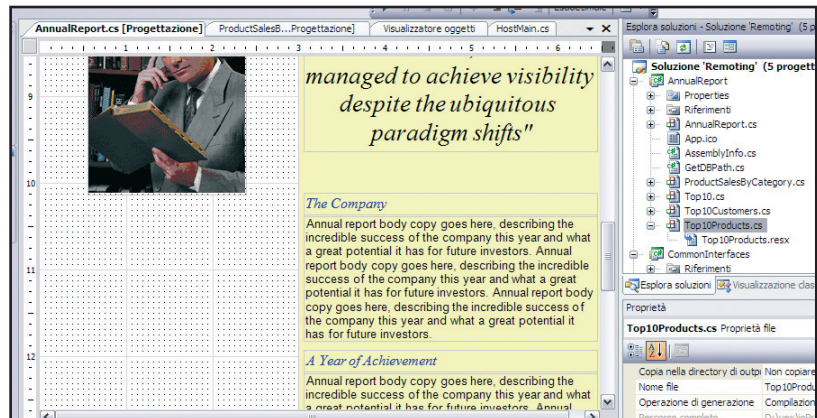



Fig. 2: Il designer di ActiveReports .NET 3.0 all'interno di Visual C# 2005 Express

```
    if (_sessions.ContainsKey(username.ToLower()))
    {
      _sessions[username.ToLower()].LastAccessTime =
                                          DateTime.Now;
      return _sessions[username.ToLower()].SessionId;
    }
    else
    {
      Session session = new Session();
      session.SessionId = Guid.NewGuid();
      session.Username = username;
      session.StartTime = DateTime.Now;
      session.LastAccessTime = DateTime.Now;

      _sessions[username.ToLower()] = session;
      return session.SessionId;
    }
  }
  return Guid.Empty;
}
```

Ed infine il metodo ProcessReport, che è quello che tira la carretta, cioè fa il vero e proprio lavoro di elaborazione del report e di restituzione del risultato finale:

```
public System.IO.Stream ProcessReport(Guid sessionId,
            System.Collections.IDictionary parameters)
{
  Session session = _getSession(sessionId);
```

```
    if (session == null) return null;

    AnnualReport rep = new AnnualReport(this,
                                        parameters);
    rep.Run(true);
    MemoryStream stream = new MemoryStream();
    rep.Document.Save(stream);

    return stream;
}
```

Specularmente il cliente fruisce di questi servizi del server:

```
private void cmdLogin_Click(object sender, EventArgs e)
{

    IRemoteServer srv = (IRemoteServer)Activator.
                            GetObject(typeof(IRemoteServer),
      "tcp://localhost:8737/remotingData/myFirstService");

    _sessionId = srv.Login(txtUsername.Text,
                                     txtPassword.Text);

    if (_sessionId != Guid.Empty)
    {
        cmdReport.Enabled = true;
        cmdRequestLock.Enabled = true;
        cmdRemoveLock.Enabled = true;
        cmdBreakableReport.Enabled = true;
    }
}

private void cmdReport_Click(object sender, EventArgs e)
{
    IRemoteServer srv = (IRemoteServer)Activator.
                            GetObject(typeof(IRemoteServer),
        "tcp://localhost:8737/remotingData/myFirst
                                              Service");
    IDictionary cfg = new Hashtable();
    cfg["parametro1"] = 13;
    cfg["parametro2"] = "Titolo per ioProgrammo";
    Stream stream = srv.ProcessReport(_sessionId, cfg);
    frmReportPreview preview = new
                              frmReportPreview(stream, cfg);
    preview.Show();
}

private void CopyStream(Stream input, Stream output)
{
    input.Position = 0;
    byte[] bytes = new byte[4096];

    int i;
    while ((i = input.Read(bytes, 0, bytes.Length)) != 0)
    {
        output.Write(bytes, 0, i);0
    }
    output.Position = 0;
}

public frmReportPreview(Stream stream, IDictionary
                                           parameters)
{
    InitializeComponent();

    MemoryStream newStream = new MemoryStream();
    CopyStream(stream, newStream);

    viewer1.Document.Load(newStream);
    viewer1.Show();
}
```

EFFETTUARE UNA CHIAMATA ASINCRONA

Partiamo da quanto già visto per risolvere il problema della sincronizzazione tra client e server in modo, ad esempio, da poter pilotare via client l'interruzione dell'elaborazione del report in corso sul server. Quello che è certo è che, seppure riuscissimo a trovare una soluzione al problema, certamente questa implicherà un livello di complessità leggermente superiore alla classica soluzione con le chiamate in locale. Il primo problema è la sincronicità della chiamata; in pratica, se il client, al momento dell'invocazione del metodo ProcessReport, resta completamente congelato in attesa che il server restituisca il controllo, è praticamente impossibile che un utente possa comandare alcunché al server o anche al client stesso visto che l'interfaccia utente non sarebbe responsiva per tutto il tempo di esecuzione della chiamata. Infatti, per la soluzione al nostro problema, il metodo ProcessReport non viene invocato direttamente, ma viene effettuata un'invocazione asincrona usando i delegati .NET. Per intenderci: ogni volta che si invoca un metodo di classe o anche una procedura o una funzione, viene restituito il controllo al chiamante solo dopo che la funzione è stata interamente eseguita, producendo in sostanza un blocco vero e proprio del chiamante fino al termine dell'esecuzione della chiamata. Fin qui nulla di nuovo, ma a volte può sorgere l'esigenza, dovendo invocare una funziona dalla lunga elaborazione, di permettere comunque al resto dell'applicazione chiamante di restare viva e responsiva ad eventuali altre richieste dell'utente. Per far ciò si ha la necessità di una chiamata asincrona e cioè di invocare la chiamata ed ottenere immediatamente la restituzione del controllo del flusso di esecuzione. Soltanto quando il chiamato avrà terminato la sua esecuzio-

ne, in qualche modo avvertirà il chiamante di aver completato l'opera così che questi possa procedere di conseguenza, magari procedendo a "consumare" il valore di ritorno della funzione chiamata, ma nel frattempo il chiamante avrà comunque potuto continuare la sua esecuzione senza blocchi o interruzioni. Questa possibilità è fornita, come si diceva, con i delegati .NET che sono fondamentalmente dei puntatori a funzione. Osserviamo il delegato per l'invocazione del ben noto metodo ProcessReport:

```
public delegate Stream ProcessReport(Guid sessionId,
                              IDictionary parameters);
```

Si noterà che ha una firma identica a quella del metodo che intende puntare. Possiamo, dunque, procedere all'invocazione del metodo via delegato. Intanto osserviamo la modalità tradizionale di invocazione:

```
IRemoteServer srv = (IRemoteServer)Activator.
                    GetObject(typeof(IRemoteServer),
      "tcp://localhost:8737/remotingData/myFirstService");
Stream stream = srv.ProcessReport(Guid.Empty,
                                  new Dictionary());
```

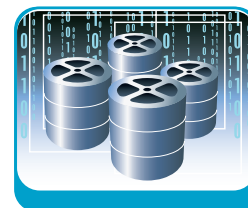
E come si trasforma passando per il delegato:

```
IRemoteServer srv = (IRemoteServer)Activator.
                    GetObject(typeof(IRemoteServer),
      "tcp://localhost:8737/remotingData/myFirstService");
ProcessReport repDel = new ProcessReport
                             (srv.ProcessReport);
IAsyncResult asyncRes = repDel.BeginInvoke
                        (_sessionId, cfg, null, null);
while (!asyncRes.IsCompleted)
{
   System.Windows.Forms.Application.DoEvents();
   Thread.Sleep(200);
   System.Windows.Forms.Application.DoEvents();
}
Stream stream = (Stream)repDel.EndInvoke(asyncRes);
```

L'istanziazione del delegato avviene passando al suo costruttore un riferimento al metodo da puntare. A quel punto si procede con l'invocazione asincrona vera e propria, richiamando il metodo BeginInvoke del delegato che ha una struttura identica al metodo puntato, con l'aggiunta di due parametri aggiuntivi sul cui significato sorvoleremo perché non è oggetto di questa trattazione. Il valore di ritorno della BeginInvoke viene assegnato ad un oggetto di tipo IAsyncResult, che ha proprio lo scopo di trattare le chiamate asincrone. A questo punto, in teoria, potrebbe concludersi il codice di chiamata e potrebbe essere restituito il controllo all'applicazione client e, soprattutto, all'utente ricevendo solo al termine dell'esecuzione del metodo ProcessReport la notifica di lavoro completato e l'eventuale valore di ritorno (il nostro Stream). In realtà, nel caso specifico non ci serve questo approccio, ma è sufficiente che il chiamante resti in attesa dell'esecuzione del metodo chiamato consentendo, però, nel frattempo di continuare a rispondere alle richieste dell'utente. Il blocco di codice successivo è una semplice while che testa la proprio IsCompleted dell'oggetto asincrono. Solo quando essa sarà vera significherà che la ProcessReport sul server è esaurita e uscirà dal blocco while consentendo, infine, attraverso la chiamata del metodo EndInvoke del delegato, di ricevere il valore di ritorno del metodo chiamato, che in questa caso è rappresentato dal nostro ben noto Stream di serializzazione del report elaborato. All'interno del blocco while sono presenti due chiamate, una al metodo DoEvents di Application, versione moderna dell'omonimo metodo di Visual Basic 6, che serve a dire all'interfaccia utente di completare l'esecuzione di tutti gli eventi di interfaccia prima di restituire il controllo di esecuzione; il secondo metodo è una banale Thread.Sleep che consente di congelare, per il tempo espresso dal parametro passato (in millisecondi), l'elaborazione. Questa tecnica, apparentemente banale, consente di ottenere due interessanti risultati: l'applicazione resta in vita durante tutta l'esecuzione del report, e la CPU del client non schizza al 100% durante la while e... sono tutti felici e contenti (il processore e l'utente).

INTERRUZIONE DELL'ESECUZIONE DEL REPORT

Adesso il client, mentre il server elabora il report, ha un'interfaccia responsiva e pertanto sarebbe nelle condizioni di poter comunicare al server anche eventuali interruzioni dell'esecuzione. Ma come fare a comunicare queste e altre quisquiglie? In apparenza il problema sembra presentare una complessità alta, ma è piuttosto semplice venirne a capo perché non si deve dimenticare che il client e il server sono pur sempre in costante contatto tra loro, per cui sarebbe sufficiente recepire la richiesta di interruzione del report mostrando all'utente un pulsante di stop e trasformare questo click in un messaggio per il server. Alla bisogna definiamo un oggetto di sincronizzazione che condivideremo tra client e server:

```
namespace ioProgrammo.RemotingServices
{
   public enum SyncStatus
   {
      Deactive,
```

```
        Running,
        Suspended,
        Stopped
    }

    [Serializable]
    public class SyncInfo
    {
        private Guid mSyncId;
        private Guid mSessionId;
        private SyncStatus mStatus;
        private int mStatusCode;
        private string mStatusMessage;
        private DateTime mStartupDate;
        public SyncInfo(Guid sessionId)
        {
            mSyncId = Guid.NewGuid();
            mStatus = SyncStatus.Deactive;
            mSessionId = sessionId;
        }

        public SyncInfo(Guid sessionId, Guid syncId)
        {
            mSyncId = syncId;
            mStatus = SyncStatus.Deactive;
            mSessionId = sessionId;
        }

        public Guid SessionId
        {
            get { return mSessionId; }
            set { mSessionId = value; }
        }

        public Guid SyncId
        {
            get { return mSyncId; }
            set { mSyncId = value; }
        }

        public SyncStatus Status
        {
            get { return mStatus; }
            set { mStatus = value; }
        }

        public int StatusCode
        {
            get { return mStatusCode; }
            set { mStatusCode = value; }
        }

        public string StatusMessage
        {
            get { return mStatusMessage; }
            set { mStatusMessage = value; }
        }

        public DateTime StartupDate
        {
            get { return mStartupDate; }
            set { mStartupDate = value; }
        }
    }

    [Serializable]
    public class SyncInfoCollection : Hashtable
    {
        public SyncInfoCollection() : base() {}

        public SyncInfoCollection(SerializationInfo info,
                                  StreamingContext context)
            : base(info, context) { }

        public void Add(object Key, SyncInfo sync)
        {
            base.Add(Key, sync);
        }

        public new SyncInfo this[object key]
        {
            get { return (SyncInfo)base[key]; }
            set { base[key] = value; }
        }

        public bool ContainsValue(SyncInfo value)
        {
            return base.ContainsValue(value);
        }
    }

}
```

L'oggetto SyncInfo è alla base dell'idea: il client ne crea un'istanza, la passa al server e al report di cui ha chiesto l'elaborazione, il server trattiene in modo centralizzato una copia del SyncInfo, e da quel momento in poi il client, ogni volta che vorrà notificare qualcosa al report, modificherà una delle proprietà della copia dell'oggetto conservata a livello centrale. Questa proprietà verrà consultata dal report che, quindi, sarà in grado di recepire la richiesta. L'oggetto SyncInfo contiene un identificativo univoco basato su un Guid (la proprietà SyncId), in modo che ogni comunicazione di sincronizzazione sia ben identificabile, l'id di sessione che ha originato la richiesta e cioè da quale client è partita (proprietà SessionId), lo stato in cui deve essere portato il processo da sincronizzare (la proprietà Status, di tipo SyncStatus, che può assumere i valori Deactive, Running, Suspended o Stopped) e altre proprietà di contorno che potrete osservare dal sorgente. Si è definita anche la collezione SyncInfoCollection

che permette di raccogliere più oggetti di Sync. Tale collezione verrà adoperata lato server per conservare l'elenco dei SyncInfo correntemente in vita. Ma come potrà il client comunicare al server un nuovo SyncInfo o modificarne lo stato di uno esistente? La risposta è nella solita interfaccia IRemoteSever del nostro Report Server: essa presenta, infatti, alcuni metodi per creare, accedere, modificare e rimuovere sync:

```
bool SetSync(Guid sessionId, SyncInfo sync);
SyncInfo GetSync(Guid sessionId, Guid syncId);
bool RemoveSync(Guid sessionId, Guid syncId);
```

Osserviamone l'implementazione fatta nel nostro solito Report Server:

```
public bool SetSync(Guid sessionId, SyncInfo sync)
{
   Session session = _getSession(sessionId);
   if (session == null) return false;

   sync.SessionId = sessionId;

   if (!mSessionLocks.ContainsKey(sessionId))
      mSessionLocks.Add(sessionId, new SyncInfo
                                        Collection());
   if
      (((SyncInfoCollection)mSessionLocks[sessionId]).
                              ContainsKey(sync.SyncId))
   {
      ((SyncInfoCollection)mSessionLocks[sessionId]).
                                   Remove(sync.SyncId);
((SyncInfoCollection)mSessionLocks[sessionId]).
                              Add(sync.SyncId, sync);
   }
   else
   {
((SyncInfoCollection)mSessionLocks[sessionId]).
                              Add(sync.SyncId, sync);
   }
   return true;

}

public SyncInfo GetSync(Guid sessionId, Guid syncId)
{
   Session session = _getSession(sessionId);
   if (session == null) return null;

   if (mSessionLocks.ContainsKey(sessionId))
   {
      if
         (((SyncInfoCollection)mSessionLocks[sessionId]).
                                   ContainsKey(syncId))
         return ((SyncInfoCollection)mSessionLocks
                                   [sessionId])[syncId];
   }
```

```

   return null;
}

public bool RemoveSync(Guid sessionId, Guid syncId)
{
   Session session = _getSession(sessionId);
   if (session == null) return false;

   if (mSessionLocks.ContainsKey(sessionId))
      if
         (((SyncInfoCollection)mSessionLocks[sessionId]).
                                   ContainsKey(syncId))
      {
((SyncInfoCollection)mSessionLocks[sessionId]).
                                        Remove(syncId);
         return true;
      }

   return false;
}
```

Il server conserva l'elenco di tutti i SyncInfo richiesti dai client nella collezione mSessionLocks. Con il metodo SetSync, a cui si passa l'id di sessione del client che ha originato la chiamata e l'oggetto SyncInfo vero e proprio, si imposta un nuovo sync. Il metodo funziona secondo il tradizionale metodo test and set: viene verificata l'esistenza di un sync con lo stesso SyncId dell'oggetto passato dal client: se è esiste, viene semplicemente sostituito con quello nuovo appena giunto, altrimenti viene creato. Con il metoodo GetSync, invece, è possibile recuperare un sinc conservato nel server ed identificato dalla proprietà SyncId passata come parametro. Infine la RemoveSync fa quel che dice...

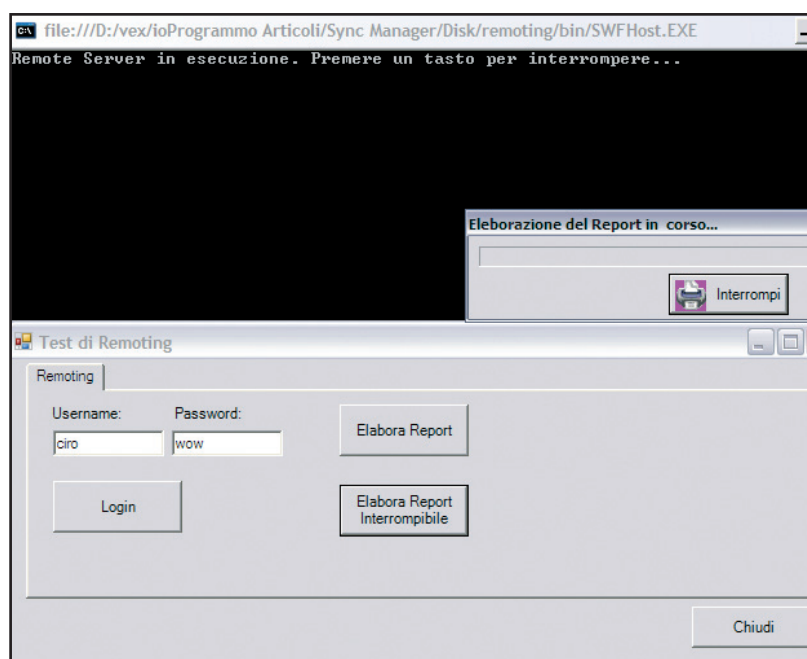
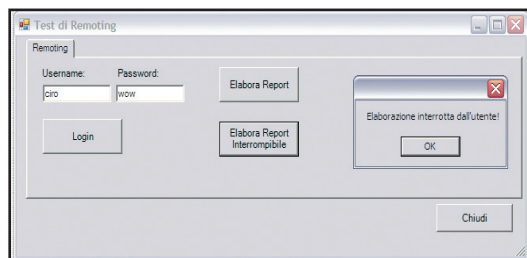



Fig. 4: Fase di elaborazione interrotta dall'utente

Fig. 5: Il report elaborato

UN ESEMPIO CONCRETO DI USO DEL SYNCINFO

Ma come avviene concretamente la richiesta di blocco di elaborazione del report da parte dell'utente? In Figura 3 possiamo osservare cosa accade quando l'utente avvia un report dal bottone "Elabora Report Interrompibile".

Si può infatti notare che, all'avvio del report appare una piccola dialog che ci avverte dell'elaborazione in corso. Al suo interno campeggia il bottone "Interrompi" che, se premuto, notifica l'interruzione al server con il sistema dei sync appena introdotto. In Figura 4 possiamo osservare cosa accade quando viene invocata l'interruzione. In Figura 5, infine, è osservabile la finestra di Preview del report che appare al termine dell'elaborazione, se questa non viene interrotta.

Ed ecco, finalmente, il codice di interfaccia del client di test che permette di avviare, ed eventualmente interrompere, l'elaborazione del report. Si tratta di una versione leggermente più sofisticata di quanto già osservato nel metodo cmdReport_Click() in precedenza:

```
//evento click del bottone che avvia il report interrompibile
private void cmdBreakableReport_Click(object sender,
                                              EventArgs e)
{
    IRemoteServer srv = (IRemoteServer)Activator.
                         GetObject(typeof(IRemoteServer),
        "tcp://localhost:8737/remotingData/myFirst
                                              Service");

    IDictionary cfg = new Hashtable();
    ProcessReportSync repDel = new
                   ProcessReportSync(srv.ProcessReport);
    SyncInfo sync = new SyncInfo(_sessionId);
    srv.SetSync(_sessionId, sync);
    IAsyncResult asyncRes = repDel.BeginInvoke
                 (_sessionId, cfg, sync.SyncId, null, null);
    frmWait frm = new frmWait(_sessionId, sync, srv);
    frm.Show();
    while (!asyncRes.IsCompleted)
    {
      System.Windows.Forms.Application.DoEvents();
      Thread.Sleep(200);
      System.Windows.Forms.Application.DoEvents();
    }
    Stream stream = (Stream)repDel.EndInvoke
                                              (asyncRes);
    frm.Close();
    if (stream != null)
    {
      stream.Position = 0;
      frmReportPreview preview = new frmReport
                               Preview(stream, cfg);
      preview.Show();
    }
}

//evento click del bottone della dialog "Elaborazione in
                    corso" che interrompe l'elaborazione
private void cmdStopReport_Click(object sender,
                                  System.EventArgs e) {
        StopReport = true;
        _sync.Status = SyncStatus.Stopped;
        _server.SetSync(_sessionId, _sync);
        Application.DoEvents();
}
```

Si può osservare che questa volta non viene invocato il vecchio metodo ProcessReport, ma un suo overload che accetta il parametro aggiuntivo syncId:

```
public System.IO.Stream ProcessReport(Guid sessionId,
                                    System.Collections.
IDictionary parameters, Guid syncId)
{
    Session session = _getSession(sessionId);
    if (session == null) return null;

    SyncInfo sync = GetSync(sessionId, syncId);
    AnnualReport rep = new AnnualReport(this,
                        parameters, sessionId, syncId);
    rep.Run(true);

    sync = GetSync(sessionId, syncId);
    if (sync != null && sync.Status == SyncStatus.
                                              Stopped)
    {
```

```
        return null;
    }
    else
    {
        MemoryStream stream = new MemoryStream();
        rep.Document.Save(stream);
        return stream;
    }
}
```

Grazie a tale parametro, il server è in grado di recuperare la copia del sync conservata a livello centrale nella collezione di tutti i sync e di passarla al costruttore report vero e proprio così che questi possa recepire l'eventuale richiesta di blocco che possa giungere dal client durante la sua elaborazione. Ed ecco, infine, cosa avviene all'interno del report. A partire dal costruttore:

```
//variabili membro private del report
IRemoteServer _server;
IDictionary _parameters;
Guid _syncId;
Guid _sessionId;

//costruttore
public AnnualReport(IRemoteServer server, IDictionary
                    parameters, Guid sessionId, Guid syncId)
{
    InitializeComponent();

    _server = server;
    _parameters = parameters;
    _syncId = syncId;
    _sessionId = sessionId;
}
```

Il report è strutturato in diversi subreport e ce ne sono due in particolare piuttosto corposi in termini di elaborazione e cioè srptTop10 e srptProductsSale. Nell metodo ReportStart, event handler che gestisce l'evento di avvio del report, vengono istanziati i due subreport in questione e ad essi vengo ripassati gli stessi parametri ricevuti dal costruttore del report e quindi anche il syncid:

```
this.srptTop10.Report = new Top10(_server,
                    _parameters, _sessionId, _syncId);
this.srptProductSales.Report = new ProductSalesBy
          Category(_server, _parameters, _sessionId,
                                             _syncId);
```

All'interno di ciascuno dei subreport, nell'evento Detail_Format, che nell'architettura di ActiveReports rappresenta l'evento che si scatena ad ogni nuova riga del report (nel caso specifico, del subreport), viene fatto un semplice polling sullo stato del sync invocandone, con la GetSync, la copia più recente conservata dal server. Qualora il sync sul server esista e sia impostato su Stopped, il report si arresta. Questo controllo viene effettuato ad nuova nuova riga di report generato, così se l'utente ha premuto Stop, il report lo saprà alla generazione delle riga successiva. In realtà, per questioni prestazionali, nell'esempio il controllo è effettuato ogni 5 righe, pertanto ci sarà una piccola latenza tra il click di stop dell'utente e l'effettivo arresto del report, latenza ulteriormente riducibile alla bisogna. Ecco il codice delle Detail_Format():

```
private void Detail_Format(object sender, System.
                                   EventArgs eArgs)
{
    if (this._iRow % 5 == 0)
    {
        SyncInfo sync = _server.GetSync(_sessionId, _
                                          syncId);
        if (sync != null && sync.Status == SyncStatus.
                                          Stopped)
        {
            this.Stop();
        }
    }
}
```

CONCLUSIONI

Partendo dal mini sistema di reportistica introdotto nel numero precedente e riepilogato in apertura di questo articolo, abbiamo risolto anche il problema del ridotto controllo che ha il client e l'utente sull'applicazione durante l'elaborazione di lunghi processi lato server come possono essere i report, introducendo una semplice architettura di sincronizzazione e di scambio di informazioni e direttive tra client e server. Nell'esempio allegato questo sistema viene adoperato per interrompere l'esecuzione di un report, ma nulla ci vieta che possa essere impiegato anche in maniera più sofisticata, magari per notificare al client lo stato di avanzamento dell'elaborazione (la classica progress bar) o una descrizione di quanto in corso di elaborazione in quel momento, informazione che si potrebbe mostrare all'utente durante l'elaborazione.
Quello che conta, infatti, è l'idea alla base e, soprattutto, l'aver dimostrato che un'architettura distribuita, seppur più complessa di una tradizionale, non pone limitazioni all'utente e allo sviluppatore

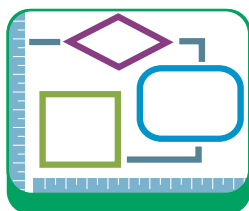
Vito Vessia

L'AUTORE

Vito Vessia progetta e sviluppa applicazioni e framework in .NET, COM(+) e Delphi occupandosi degli aspetti architetturali. Scrive da anni per le principali riviste italiane di programmazione ed è autore del libro "Programmare il cellulare", Hoepli, 2002, sulla programmazione dei telefoni cellulari connessi al PC con protocollo standard AT+. Può essere contattato tramite e-mail all'indirizzo vvessia@katamail.com.

PROXY: MENTIRE A FIN DI BENE

VOLETE OTTIMIZZARE LE PRESTAZIONI DEL VOSTRO SISTEMA? O MAGARI DOVETE GESTIRE LA SICUREZZA, O ACCEDERE AD OGGETTI REMOTI? IN TUTTI QUESTI CASI IL PATTERN PROXY POTREBBE ESSERE LA SOLUZIONE.

CD WEB
proxy_codice.zip
cdrom.ioprogrammo.it

Il capo è stato fin troppo chiaro: "Stokko", il nostro favoloso servizio Web 2.0 per seguire l'andamento dei titoli di borsa, deve essere in produzione entro domani, già integrato con i nostri servizi di analisi finanziaria. Ecco i requisiti:

- *Un utente si connette al sito con un browser o un cellulare e vede una serie di nomi di titoli ("MST" per Microsoft, "SUN" per Sun, eccetera).*
- *L'utente seleziona il titolo che vuole esaminare, e nella pagina appare un grafico che gli mostra l'andamento del titolo nelle ultime ventiquattr'ore.*

Il fatto è che il sistema è già quasi pronto, ma ha un problema: è lento come la deriva dei continenti. Il collo di bottiglia è nella creazione degli oggetti che rappresentano i grafici. Per essere sempre aggiornato, il server ricostruisce tutti i grafici ogni venti secondi, anche se nessuno ha richiesto quel particolare grafico. Calcolare un grafico è un'operazione costosa, che richiede memoria per conservare l'immagine e alcuni accessi ad un database. Sarebbe molto meglio fare queste operazioni solo quando sono indispensabili, cioé calcolare solo i (pochi) grafici che vengono chiesti dagli utenti.
"Poco male", pensate voi - "Basterà cambiare un po' il design". Ecco la classe che rappresenta un singolo titolo:

```
class Stock:
    def __init__(self, identifier):
        self.__load(identifier)

    def __load(self, identifier):
        # lenta e pesante inizializzazione
        import time
        time.sleep(3)

    def draw(self):
        print "->Stock.draw()"
```

CENNI DI PYTHON

Questo codice è scritto in Python, ma dovrebbe essere abbastanza comprensibile anche per chi non conosce questo linguaggio. È la definizione di una classe di nome *Stock*, che contiene tre metodi.
Una caratteristica di Python che potrebbe stupire chi non lo conosce è che non esistono parentesi per delimitare i blocchi di codice – si usa semplicemente l'indentazione. Non fate caso all'oggetto *self* che appare un po' dappertutto – Python ci impone di dire sempre a quale oggetto facciamo riferimento, anche se quell'oggetto siamo noi stessi. La parola *self* appare anche come primo argomento di tutti i metodi, e anche in questo caso potete ignorarla. Quando chiamate il metodo, non dovete passare un valore a questo parametro – il sistema ci pensa da solo. Lo sappiamo, è strano, ma è una delle poche caratteristiche di Python che non sono molto intuitive.

COSTRUTTORI

Il metodo *__init__* è l'equivalente Python di un costruttore – viene chiamato automaticamente quando si crea un nuovo oggetto. Il costruttore prende come parametro l'identificativo del titolo in borsa e lo usa per cercare le informazioni nel database e generare il grafico. Tutte queste operazioni lente e complicate sono eseguite dal metodo *__load* (i due underscore davanti a questo nome sono una convenzione per indicare che il metodo è privato). Per brevità, in questo esempio abbiamo simulato la generazione del grafico con una semplice pausa di tre secondi. Infine, il metodo *draw* restituisce il grafico da spedire al

browser del client. Per simularlo abbiamo usato una semplice stampa.

UNA SOLUZIONE NON OTTIMIZZATA

Per creare uno di questi oggetti servono dunque tre secondi. Troppi, se dobbiamo crearne centinaia ogni venti secondi. Possiamo risolvere il problema modificando il codice della classe *Stock* per caricare il grafico solo quando qualcuno chiama *draw*, ma nel nostro caso questa non sarebbe una soluzione elegante: questo codice fa parte del nucleo del sistema, che viene usato da diverse applicazioni e gestito da un altro team. Non vogliamo andare a mettere le mani in tutti i sistemi solo per ottimizzare l'interfaccia Web. Un'altra possibilità è quella di dividere la classe in due: il grafico vero e proprio, e un descrittore che contiene solo le informazioni che possono essere calcolate velocemente. Questa soluzione, però, sfrutta poco le caratteristiche dei linguaggi a oggetti: devo sempre ricordare di convertire tra due oggetti diversi, mentre vorrei avere un riferimento ad un solo oggetto che viene passato in giro per il sistema e contiene tutte le informazioni legate ad un singolo titolo di borsa (eventualmente spezzettate in oggetti più piccoli).

UNA VERSIONE PIÙ FUNZIONALE

Esiste una soluzione più elegante. Possiamo definire una nuova classe che finge di essere uno *Stock*, pur senza esserlo. Poi posso passare in giro oggetti di questa classe - dato che gli oggetti fingono di essere dei veri *Stock*, nessuno se ne accorgerà. Quando e se non potranno farne a meno, questi oggetti potranno creare dei veri *Stock* e delegare a loro la logica dei titoli:

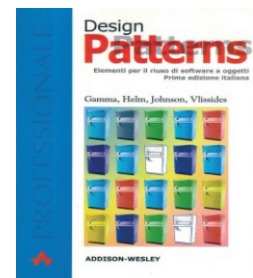
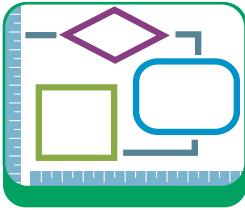
```
class StockProxy:
    def __init__(self, identifier):
        self.identifier = identifier
        self.realSubject = None

    def draw(self):
        if self.realSubject == None:
            self.realSubject = Stock(self.identifier)
        self.realSubject.draw()
```

L'interfaccia di *StockProxy* è identica a quella di *Stock*, ma il codice si comporta in modo diverso. *StockProxy* conserva l'identificativo del titolo in un campo privato. Definisce anche un riferimento di nome *realSubject* ad un vero *Stock*, che però è inizialmente nullo. Quando qualcuno chiama draw, *StockProxy* ha bisogno di un vero *Stock* per svolgere l'operazione. Quindi controlla se il suo *Stock* è già stato creato, e in caso contrario lo crea usando l'identificativo che aveva messo da parte. Poi delega il disegno del grafico al vero *Stock*. Scriviamo un piccolo test per vedere se tutto funziona:

```
stocks = []
for i in range(10000):
    stock = StockProxy("item" + str(i))
    stocks.append(stock)
stocks[100].draw()
stocks[1000].draw()
```

I pythonari storceranno un po' il naso di fronte a questo codice, perché le stesse operazioni possono essere scritte in modo più elegante e compatto – ma preferiamo che il codice sia abbastanza comprensibile anche per chi non conosce Python. La funzione *range* restituisce un intervallo di valori tra 0 e 9999, e il ciclo *for* itera su questi valori. Quindi questo codice crea un array di diecimila *StockProxy*, i cui identificativi vanno da (item0 a item999) e alla fine ne visualizza solo due. Dopo pochi

IL BELLO DEI PATTERN

I programmatori si scontrano spesso con gli stessi problemi in circostanze diverse. Per i principianti della programmazione a oggetti, ogni problema sembra nuovo. Gli esperti, invece, hanno già sviluppato quel sesto senso che permette loro di dire: "Ho già incontrato un problema simile prima, e ho una buona soluzione". Sarebbe bello che i principianti potessero riconoscere gli stessi problemi senza passare attraverso un lungo percorso di tentativi ed errori. I pattern sono la soluzione.
L'idea dei pattern non è quella di dare ricette da applicare copiando il codice, ma quella di presentare delle soluzioni-tipo che ciascuno può applicare in modo diverso ai problemi che incontra. Una delle caratteristiche più importanti dei pattern è il nome. Se io e il mio collega conosciamo il pattern Proxy, posso dirgli semplicemente "questa classe è un Proxy di quest'altra", e lui capirà al volo cosa intendo dire. Molto meglio che spiegare ogni volta tutte le complicate interazioni tra le classi nel nostro software.
Nel 1995, il libro "Design Patterns" cambiò il panorama dell'informatica. Il libro era un catalogo di ventiquattro pattern comuni, incluso Proxy. Da quel momento in poi, il concetto di pattern (che deriva dall'architettura e dall'urbanistica) è entrato a far parte del vocabolario comune dell'ingegneria del software. Anche se il libro non è di facile lettura, e se la gran parte degli esempi sono scritti in C++, vale comunque la pena di leggerlo.

istanti necessari per creare tutti gli *StockProxy*, e tre secondi aggiuntivi per ciascuno dei due *Stock* che effettivamente ci servono, il programma termina con questo risultato:

```
->Stock.draw
->Stock.draw
```

Fatto! Il capo sarà entusiasta.

IL PATTERN PROXY

Quello che abbiamo visto è un pattern molto comune che si chiama *Proxy*. Un *Proxy* è un guscio sottile intorno ad un oggetto. Possiamo chiamare questo oggetto *Real Subject*, "soggetto reale". Il *Proxy* finge di essere il Real Subject, in modo che il client non noti la differenza (parliamo di "client" nel senso della programmazione a oggetti, cioè "qualsiasi pezzo di codice che accede all'oggetto", tipicamente un altro oggetto). Il Proxy delega al Real Subject le operazioni complicate – ma non senza metterci qualcosa di suo. In questo caso, ad esempio, il Proxy serve ad ottimizzare delle operazioni lente, deferendo la creazione del Real Subject al momento in cui proprio non se ne può più fare a meno. Usato in questo modo, il Proxy si chiama di solito *proxy virtuale*.

PROXY ON THE ROAD

Esistono anche altri problemi che si possono risolvere con un Proxy. Uno dei più comuni è quello degli oggetti distribuiti. Il mio codice client vorrebbe manipolare degli oggetti locali, ma in realtà questi oggetti risiedono su qualche altra macchina. Il client non vuole preoccuparsi dei dettagli legati alle comunicazioni – vorrebbe solo usare gli oggetti. Quindi possiamo scrivere dei Proxy che si comportano come normali oggetti, e per ogni chiamata gestiscono la comunicazione con i veri oggetti attraverso la rete. In quasi tutti i framework per oggetti distribuiti (come gli EJB di Java) esiste qualcosa di simile. Questo tipo di Proxy si chiama *proxy remoto*.

I Proxy possono essere utili anche per questioni di sicurezza. Un *proxy di protezione* controlla se il client ha il diritto di eseguire una certa operazione. Se sì, il Proxy delega al Real Subject; se no, ignora il comando o genera un errore.

Un'altra situazione nella quale il proxy può essere utile è la cosiddetta *copy-on-write*. A volte capita di avere degli oggetti che sono particolarmente costosi da copiare (per esempio grosse strutture in memoria) – ma siamo costretti a copiarli, perché alcuni dei client vogliono una propria copia per modificarla. Una possibile soluzione è copiare l'oggetto il più tardi possibile. Ogni volta che il client vuole la propria copia dell'oggetto, gli restituiamo in realtà l'oggetto originale avvolto in un Proxy. Quando e se il client modifica l'oggetto, il Proxy si occupa di farne una copia. In questo modo la copia avviene solo se ce n'è bisogno.

Nei linguaggi come il C++, che richiedono una gestione manuale della memoria, i Proxy possono tornare molto utili. Gli *smart pointer* del C++ sono Proxy che contano i riferimenti agli oggetti, per essere certi che tutta la memoria allocata sia liberata al momento giusto. Come per tutti i pattern, potete incontrare molti altri problemi che si risolvono bene con un Proxy.

PROXY DA ALTRI MONDI

Il termine "proxy" non si usa solo nel campo della programmazione a oggetti. Sicuramente avete già incontrato il concetto di "proxy di rete". È un software che finge di essere un server di rete, ma intercetta le chiamate al vero server e le filtra nel caso non siano legittime, oppure mantiene alcuni risultati in cache per ottimizzare gli accessi al network. Quindi un proxy di rete funge sia da "proxy di sicurezza" che da "proxy virtuale", due usi comuni anche per l'omonimo pattern. In generale, la parola "proxy" in inglese indica qualcuno che agisce in nome e per conto di qualcun altro.

PROXY INTELLIGENTI

Lo *StockProxy* del nostro esempio delega la sua unica operazione (*draw*) al Real Subject. Non sempre vogliamo un Proxy così pigro. A volte possiamo risolvere alcune operazioni molto semplici direttamente nel Proxy, senza creare il Real Subject. L'idea è di mettere nel Proxy solo le operazioni banali - non vogliamo che il Proxy entri in concorrenza con il Real Subject. Ad esempio, la classe *Stock* potrebbe avere un metodo *info* che restituisce il suo identificatore:

```
class Stock:
    def __init__(self, identifier):
        self.identifier = identifier
        self.__load(identifier)
```

```
    def info(self):
        return self.identifier

    def draw(self):
        print self.info() + ".draw()"
...
```

Anche lo *StockProxy* deve avere la sua implementazione di *info*, se non vogliamo ricevere un errore ogni volta che un client chiama questo metodo sul Proxy anziché sul Real Subject. Quindi potremmo pensare che anche in questo caso lo *StockProxy* deve creare uno *Stock* per poi delegargli il metodo *info*. In realtà, *StockProxy* conserva già l'identificativo del titolo, quindi può dare questa semplice informazione anche senza creare uno Stock:

```
class StockProxy:
    def info(self):
        return self.identifier
...
```

Ora possiamo stampare informazioni sui titoli senza compromettere le scattanti prestazioni a cui abbiamo abituato quel viziato del capo:

```
stocks = []
for i in range(10000):
    stock = StockProxy("item" + str(i))
    stocks.append(stock)
    print stock.info()
stocks[100].draw()
stocks[1000].draw()
```

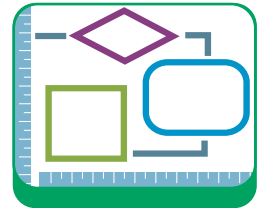
Questo è solo un esempio di come potete adattare il pattern a circostanze particolari. Non pensate mai ai pattern come a delle ricette da seguire pedissequamente: di ciascun pattern esistono infinite varianti, e sta a voi decidere come e quando usarle.

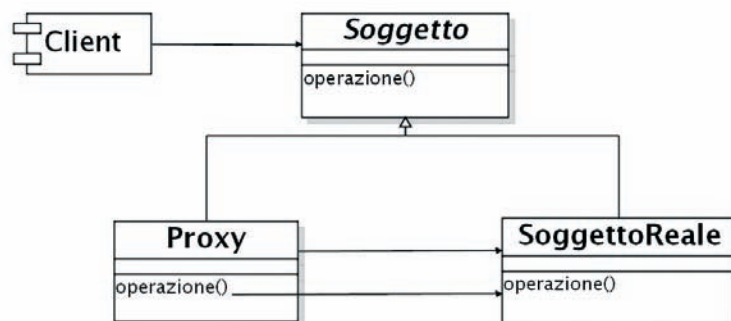
CONCLUSIONI

Come tante cose nell'informatica, i Proxy hanno un vantaggio e un costo. Il vantaggio è che fingono di essere il vero oggetto, e ci permettono di controllare in modo trasparente l'accesso all'oggetto. Lo svantaggio è un problema di identità. In realtà il Proxy non è l'oggetto che dice di essere, e questa piccola bugia a fin di bene diventa pericolosa se confrontate Proxy e oggetti tra loro. Nei linguaggi che lo permettono, potrebbe avere senso ridefinire l'operatore di uguaglianza, ma qui ci addentriamo su un terreno potenzialmente pericoloso.

Come facciamo a capire se ci serve un Proxy? Come abbiamo appena detto, pensatelo come un pattern per controllare l'accesso ad un oggetto. Ogni volta che questo accesso deve sottostare a delle regole, pensate se potete semplificarvi la vita introducendo un intermediario tra client e oggetto. Questo intermediario è il Proxy.

Ovviamente ciascuna cosa deve essere presa con moderazione, anche i proxy fanno parte di quella categoria di strumenti dei quali non si deve abusare. E' importante utilizzare il proxy nei contesti a cui si rferisce. Per ogni approfondimento è possibile fare riferimento alla bibliografia riportata nell'articolo ed ovviamente al forum di ioProgrammo http://forum.ioprogrammo.it

Paolo Perrotta

TIPI STRANI

Gli esempi di questo articolo sono scritti in Python, un linguaggio dinamicamente tipato. Se usate un linguaggio staticamente tipato, come Java o C#, implementare un proxy è leggermente più difficile. In questo caso, infatti, il compilatore controllerà con la sua proverbiale pignoleria che tutti i tipi dichiarati nel vostro sistema siano coerenti. Questo significa che se volete sostituire un SoggettoReale con un Proxy, i due oggetti devono avere lo stesso tipo, cioè estendere la stessa classe o implementare la stessa interfaccia. In linguaggi come Python o Ruby non avete questo problema: tutto quello che serve è che il Proxy e il SoggettoReale siano in grado di ricevere gli stessi "messaggi" (cioè che abbiano metodi con lo stesso nome e argomenti). Non è un grosso problema soddisfare le esigenze del compilatore. Se il SoggettoReale e il Proxy implementano la stessa interfaccia (o ereditano dalla stessa classe), è possibile sostituire il primo al secondo: Naturalmente, come sempre nei pattern, potete inventare diverse variazioni sul tema. Ad esempio, la classe del Proxy può ereditare direttamente dalla classe del SoggettoReale.
Un buon esempio di Proxy in Java sono le interfacce locali degli Enterprise JavaBeans. Queste interfacce sono dei Proxy remoti che convogliano le chiamate del client ai SoggettiReali (i componenti EJB), che di solito risiedono su un altro computer. Il client ha l'illusione che l'EJB sia accanto a lui, nella stessa macchina virtuale.

SOFTWARE SUL CD

STRUTS 2.0

IL FRAMEWORK JAVA PER IL PATTERN MVC

Il Model View Controller, è uno dei pattern maggiormente utilizzati in programmazione. Il suo scopo è semplice, ovvero dividere la struttura di un programma in tre moduli separati. Uno che si occupa della definizione delle classi di business ovvero il model, uno che si occupa di visualizzare i dati in output, ovvero la view, uno che si occupa di gestire il flusso dell'esecuzione. Un software che segue i dettami dell'MVC può dirsi ben strutturato, solido e facilmente manutenibile. Ora, mentre il pattern è unico, possono essistere diverse soluzioni applicative per la sua implementazione e non tutte efficaci. Struts mette a disposizione un framework ben strutturato ed ormai consolidato per la creazione di programmi Java aderenti al pattern MVC. Nel tempo è diventato un vero e proprioi standard e pur conservando una certa complessità nessun programmatore Java che voglia sviluppare software professionale può fare a meno di aderirvi completamente.

Directory: /struts

DAYPILOT 2.1. SP3

UN CONTROLLO ASP.NET PER LA GESTIONE DEGLI APPUNTAMENTI

Intelligente questo controllo! Consente di creare un'applicazione ASP .NET che offre funzionalità del tutto simili a quelle utilizzate da Outlook nella sua parte relativa alla gestione degli impegni.Il controllo è molto solido e ben strutturato e offre una serie di funzioni che risultano piuttosto comode per un programmatore, molto più evolute di un normale calendario

Directory: /daypilot

ZENCART 1.3.7

IL TUO NEGOZIO ONLINE IN 5 MINUTI

Si tratta di una soluzione completa per il commercio elettronico. Ricca di funzionalità e di gateway di pagamento offre il supporto per un numero illimitato di prodotti e categorie, funzionalità per la gestione delle promozioni e altri extra come la gestione dei banner e delle newsletter. In definitva si tratta di un software per l'e-commerce completo e funzionale

Directory:/ ZenCart

NHIBERNATE 1.2.0 BETA

COME HIBERNATE MA PER .NET

Neanche .NET sfugge alla logica secondo cui i database relazionali sono difficilmente rappresentabili come oggetti. Ed ecco che arriva NHibernate che seguendo lo stesso paradgima di Hibernate garantisce il mapping fra oggetti ed elementi di un database relazionale superando i limiti imposti dalle due diverse tecnologie.Il tool è particolarmente interessante, sia per l'elevato gradodi disaccoppiamento che offre tra glielementi del database e le classi chelo gestiscono, sia per la capacità ditrasformare dati tipicamente trattati in modo relazionale in classi ed oggetti

Directory: /Nhibernate

MYSQL 5.0.27

IL PRINCIPE DEI DATABASE

Indispensabile per programmare webapplication in tecnologia PHP. Nonche non sia possibile utilizzare altridatabase, ma MySQL e PHP rappresentano veramente un binomioinscindibile. L'integrazione fra questo database e il linguaggio di scripting più usato sulla rete è talmente alta da fare divenire quasi un obbligo l'uso congiunto di questi due strumenti

Directory:/MySQL

ACQUA DATA STUDIO

IL SQL MANAGER È SERVITO

Acqua Data Studio è uno strumento decisamente evoluto. Scritto in Java si configura come un SQL Manager completo e affidabile. Supporta un gran numero di DBServer, da Oracle a MsSQL a MySQL. Consente la gestione totalmente grafica delle tabelle, dei database, delle interrogazioni SQL. Si può usare in alternativa a PHPMyAdmin anche se è ancora leggermente inferiore per quanto riguarda il numero di funzionalitàesposte. D'altra parte PHPMy-Admin è un fuoriclasse nel suo genere ed è tarato su MySQL, mentre Acqua-Data Studio è del tutto generico. In ogni caso si tratta di un sistema che vi può trarre fuori dai guai in più di una situazione. Da provare!

Directory:/ aquadatastudio

POSTGRES 8.2.1

IL GRATIS COMPLETO E VELOCE

Nessun server di database offre la completezza delle funzioni esposte da PostgreSQL e rimane comunque completamente Gratis. PostgreSQL è probabilmente il più estendibile fra i database esistenti. Inutileparlare della gamma praticamente completa delle sue funzioni. Il punto di forza che lo pone probabilmente al di sopra di tutti i concorrenti rimane l'alta possibilità di personalizzazione, oltre, naturalmente, alla velocità, alla stabilità ed al costo nullo.Unica pecca, una certa complessità nella gestione. E' sicuramente da usare in ambienti di produzione professionali che non possono accontentarsi di alcun compromesso

Directory:/ postgres

FIREBIRD 2.0.0.12748-0

IL DATABASE VELOCE COME UN FULMINE

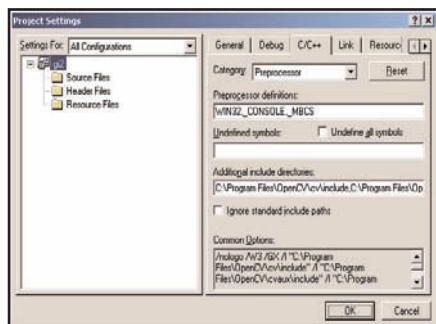
Ha attraversato una serie incredibile di disavventure. Acquistato da Borland è poi tornato ad essere OpenSource. Presente sul mercato da tempo immemorabile è riuscito a sopravvivere alle sue varie vicissitudine solo grazie alle sue grandi doti tecniche. E' un database velocissimo e ultraleggero, dotato però di tutte le funzioni di un server professional

Directory:/ Firebird 2.0.0.12748-0

OPEN COMPUTER VISION LIBRARY 1.0

OTTIMO PER LA GRAFICA

Più che una semplice libreria per la gestione degli oggetti grafici. Si tratta di un'insieme di API piuttosto futuristiche, si va da riconoscimento del movimento a quello facciale, all'isolamento dei contorni. Si tratta di una libreria immensa che offre un quantitativo sterminato di funzioni relative al multimedia e alla grafica

Directory: /opencv

SPHINX 4.0.1 BETA

IL RICONOSCITORE DI PAROLE

Abbiamo già parlato di FreeTTScome un motore in grado di riprodurreun testo e trasformarlo in unoutput vocale. Non abbiamo dettoperò qual'è il tool che consente di riconoscere una parola, stabilire il giusto accento e la giusta pronuncia,questo tool è Sphinx4.La versione presentata è la 1.0 beta,non tutte le parole sono riconosciute in modo ottimale, il legame con lalingua italia è scarso, ma se volete iniziare a provare come funzionano tool di questo genere sicuramente èun ottimo inizio

Directory:/Sphinx

WAMP SERVER 5.1.6.6

PHP, APACHE E MYSQL IN UN COLPO SOLO

Wamp server consente di installare in modo semplificato un completo ambiente di test per le applicazioni PHP. L'intera suite si compone di Apache + PHP + MySQl e garantisce l'installazione di tutti i prodotti aggirando le complicazioni che potrebbero nascere da un'installazione manuale

Directory:/ wamp

INNOSETUP 5,1,9

IL CREATORE DI SETUP ALTERNATIVO

Non esiste solo la programmazione. Come si fa a distribuire un software una volta che la fase di sviluppo è terminata? Esistono diverse soluzioni, InnoSetup consente di creare sofisticati Wizard per l'installazione di un software, senza troppe complicazioni. Altro punto a favore è il suo costo praticamente nullo. Con InnoSetup si possono dunque creare procedure di installazione professionali a costo zero

Directory:/innosetup

ASPECT C++ 1.0 PRE 3

PROGRAMMARE AD ASPETTI CON C++

La programmazione ad Aspetti nascedall'esigenza di riunire insieme molti "spezzoni" di programmi diversi, un po' come avveniva con la programmazione procedurale ma con un legame più stretto agli eventi. Consente di definire dei Joint Points a cui viene passato il controllo in relazione ad un evento specifico.Questa libreria consente appunto di adottare questo approccio anche con software scritto in C++

Directory:/ AspectCPP

BEANSHELL 2.0.4 B

PER DOTARE LE TUE APPLICAZIONI DI UN LINGUAGGIO DI SCRIPTING

Beanshell è un linguaggio di scripting e al contempo un engine embedded per le tue applicazioniJava. Sostanzialmente si può inserire l'engine all'interno dell'applicazione e consentire agli utenti di utilizzare il linguaggio per estendere il software con moduli. In pratica una sorta di linguaggio Macro applicabile direttamente al software sviluppato da te. Molto interessante e utile in tutti quei casi in cui si vuole offrire una profonda personalizzazione senza dover ogni volta ricompilare ilcodice

Directory:/ beanshell

JAMELEON 3.3

UN FRAMEWORK AUTOMATIZZATO PER IL TESTING DELLE APPLICAZIONI

Jameleon è un tool che consente ditestare con efficacia il software da voi creato. Il concetto principale su cui Jameleon si basa è quello dei Tag che rappresentano una schermata dell'applicazione.Ciascun gruppo di Tag può essere parametrizzato con dati e chiaramente deve restituire un certo output. Infine tutto può essere automatizzato producendo degli script che salvano i risultati in file di log

Directory:/ jameleon

SPRING 2.0.2

IL FRAMEWORK DEI FRAMEWORK

Spring è un Framework che riuniscesotto un unico cappello una serie distrumenti già esistenti mettendoli ingrado di comunicare in modo correttofra loro. Inoltre Spring implementail pattern Ioc, inversion of controlche garantisce un alto grado di disaccoppiamento e una maggiore manutebilità del codice

Directory:/Spring

JAD 1.5.8

THE FAST JAVA DECOMPILER

Un utility interessante che consente di ottenere il sorgente di un software a partire dalla sua classe Java. Non è aggiornato all'ultimissima versione del compilatore ma fornisce buoni risultati con la maggior parte delle classi java oggi disponibili

Directory:/Jad

BUGZILLA 2.23.3

PER TENERE SOTTO CONTROLLO I BACHI DEL SOFTWARE

Un buon software evolve nel temposulla base delle indicazioni dei suoi utenti. E tutti i software nascono con qualche imperfezione che solitamente viene fuori proprio in fase di utilizzo intensivo dell'applicazione.Ma come tener traccia delle segnalazioni effettuate dagli utenti? Bugzilla è una Web Application checonsente di memorizzare i bug, ordinarli secondo precisi ticket e così garantisce al programmatore di poter intervenire in modo organico, sistemando i bachi, elaborando patch, eventualmente segnalando all'utente falsi bug.

Un'applicazione senza dubbio utile che risolve uno dei maggiori problemi del ciclo di sviluppo. Attualmente bugzilla è lo strumento più diffuso per il controllo dei bug sui progetti opensource e molto spesso viene utilizzato anche in caso di progetti commerciali. L'installazione in ambiente Windows richiede un po' di pazienza. Nessun problema su hosting Linux

Directory:/ bugzilla

EASYPHP 2.0B1

WEB SERVER, DATABASE E PHP TUTTO IN UN CLICK

Sono in molti coloro che vorrebbero imparare PHP e che prima ancora che nel linguaggio trovano difficoltà nell'installazione di un ambiente di test per la programmazione. Per poter testare un software scritto in PHP è necessario avere installato almeno un Web Server ed il linguaggio stesso, opzionalmente è necessario MySQL. L'installazione di questi prodotti è per molti il primo ostacolo all'apprendimento di PHP. EasyPHP è un software All in Once che con pochi click vi consentirà di installare tutto il necessario senza avere alcuna conoscenza di sistemistica

Directory: /easyphp

JAVA SE DEVELOPMENT KIT 6

IL COMPILATORE INDISPENSABILE PER PROGRAMMARE IN JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti

Directory: /java6

FIDDLER 1.2

UN PROXY DEBUGGER

Fiddler si pone in mezzo fra voi e la rete internet e analizza qualunque url passa attraverso la vostra scheda di rete. Utile se installato sopra un proxy, consente di capire come viene direzionato il traffico di rete da parte dei computer client

Directory:/ fiddler

PYTHON 2.5

L'EX GIOVANE RAMPANTE

Python è stato considerato per lungo tempo il nuovo che avanza. Attualmente non lo si può più definire in questo modo, Python è ormai un linguaggio stabile e completo che trova applicazione inun gran numero di progetti. Se ne parla sempre di più in campo industriale come su Internet. Soprattutto un gran numero di applicazioni anche in ambiente Windows girano ormai grazie a Python e presentano interfacce grafiche ottimamente strutturate. Ciò nonostante Python rimane un grande linguaggio di scripting adatto a gestire in modo completamente automatico buona parte di un sistema operativo sia esso Linux o Windows

Directory:/ Python

ECLIPSE SDK 3.2.1

L'IDE TUTTOFARE

Eclipse è un progetto completo portato avanti da Eclipse Foundation con la collaborazione di una miriade di aziende fra cui IBM, Adobe, Sun e che si è prefissata lo scopo di creare un IDE estendibile per plugin adattabile a qualunque tipo di linguaggio o tecnologia. Di default Eclipse si propone come IDE per Java ed è qui che da il meglio di se. Ma proprio grazie ai suoi plugin è possibile utilizzarlo come ambiente di programmazione per PHP, per C++, per Flex e per molti altri linguaggi ancora. Inoltre sempre grazie per ciascun linguaggio sono disponibili altri plugin ad esempio per rendere l'ambiente RAD o per favorire lo sviluppo dei Web Services o altro. Insomma lo scopo è stato raggiunto completamente. Eclipse è realmente un IDE tuttofare, ormai maturo, e che serve una miriade di programmatori grazie alle sue caratteristiche di affidabilità e flessibilità. Unica nota negativa: una certa pesantezza che lo rende idoneo ad essere usato solo su PC con una dotazione hardware minima di tutto rispetto

Directory:/ Eclipse

PHP 5.2.0

IL LINGUAGGIO DI SCRIPTING PIÙ AMATO DEL WEB

Sono tre le colonne portanti di Internet: PHP, APACHE e MySQL. Certo la concorrenza è forte. Asp.NET e SQL Server avanzano con celerità, ma a tutt'oggi non si può affermare che i siti sviluppati in PHP costituiscano la stragrande maggioranza di Internet. Quali sono le ragioni del successo del linguaggio? Prima di tutto la completezza. PHP ha di base tutto quello che serve ad un buon programmatore, raramente è necessario ricorrere a librerie esterne, e quando è proprio indispensabile farlo esistono comunque una serie di repository che rendono tutto disponibile . In secondo luogo una curva d'apprendimento praticamente nulla

Directory:/ PHP

SE BIANCO E NERO NON BASTANO...

LA LOGICA FUZZY SI SVINCOLA DALLA MERA VISIONE BINARIA DELLA SOLUZIONE DI UN PROBLEMA DA RISOLVERE. INTRODUCE UN NUOVO E PIÙ NATURALE MODO PER AFFRONTARE UNA VASTA CLASSE DI QUESTIONI ELIMINANDO LA COPPIA: VERO-FALSO

Le discipline e gli studi introdotti nell'era digitale da una parte e la consolidata filosofia aristotelica, come fondamenta della logica classica, dall'altra hanno portato ad una interpretazione molto rigida dell'universo e delle leggi che ne descrivono l'evoluzione. Alla base della moderna logica che influenza molte scienze dalla matematica alla fisica è presente un rigoroso carattere di bivalenza. Un bit o è zero o è uno; una proposizione o è vera o è falsa; un elemento o appartiene o non appartiene ad un insieme. Ma anche nel linguaggio comune si mantiene questa rigidità. Un oggetto o è freddo o è caldo; una persona o è alta o è bassa e si potrebbe così continuare all'infinito. Come risultato di questo processo, va rilevata ad ogni modo, una forte influenza della discretizzazione in molte scelte che l'uomo in questi ultimi anni ha compiuto. Un esempio per tutti è la certificazione dei saperi che avviene sempre più spesso con dei test a risposta multipla. Ciò ha portato sicuramente ad una più rapida fase di verifica ma ha anche svilito l'attestazione delle abilità dell'esaminato, in particolare le capacità di analisi, rielaborazione e sintesi, che i vecchi temi garantivano. Si assiste anche a paradossi di altri tipi. Supponiamo di classificare rigidamente l'età di una persona facendolo appartenere ad uno di questi insiemi bambino, giovane, maturo, anziano, vecchio e che la classe anziano vada dai 55 agli 80 anni. Ci troveremmo costretti a considerare una persona nel giorno del suo cinquantacinquesimo compleanno un anziano. Quindi il giorno prima questo signore era maturo e il giorno dopo si ritrova anziano. Non solo per cortesia o per evitare una crisi depressiva alla persona, si potrebbe considerare una sorta di proroga; anche di pochi mesi che ne constaterebbe ancora l'appartenenza alla classe dei maturi. Così però si raggiungerebbe il paradosso che la persona non passi mai nella classe degli anziani ne tantomeno dei vecchi. Così una persona di 100 anni potrebbe essere ancora bambino. La teoria proposta negli anni sessanta da Zadeh, la logica sfocata o sfumata, in una sola parola fuzzy, tenta di superare questa empasse proponendo di considerare i valori intermedi. Così un elemento, come vedremo, non sarà più appartenente o non appartenente ad un insieme ma ad esso sarà associato un grado di appartenenza a ciascuno dei due insiemi. Dopo aver dato uno sguardo all'evoluzione storica di questa disciplina, esamineremo le basi della logica fuzzy per poi esaminare alcuni operatori fuzzy e un esempio.

STORIA

Fu un memorabile articolo del 1965 scritto dal professore di ingegneria Lofti A. Zadeh dell'Università californiana di Berkeley a introdurre una nuova prospettiva per il trattamento si alcune classi di problemi. Il professore faceva un esempio classificando gli animali. E sebbene alcuni animali come cani, gatti e cavalli, trovavano una collocazione precisa stessa cosa non si poteva dire per altri come le stelle marine le spugne. E così propose un modello matematico dai contorni indefiniti che trattasse problemi imprecisi intrinsecamente. Ad

SOFT COMPUTING

Un nuovo modo elaborare si sta facendo strada, si tratta del soft computing – calcolo soft – che prova a trattare l'imprecisione intrinseca dei dati restituendo soluzioni "accettabili", non sempre ottime, ma che possono essere soddisfacenti. Il calcolo soft si basa su tre pilastri la programmazione genetica, le reti neurali e appunto la logica fuzzy. C'è un passaggio di Zadeh che descrive meglio di ogni altro cosa si intende per soft computing; "... il soft computing si prefigge lo scopo di adattarsi alla pervasiva imprecisione del mondo reale. Il suo principio guida può così esprimersi: sfruttare la tolleranza per l'imprecisione, l'incertezza e le verità parziali in modo da ottenere trattabilità, robustezza e soluzioni a basso costo. Nei prossimi anni, il soft computing è probabilmente destinato a giocare un ruolo sempre più rilevante nella concezione e progettazione di sistemi il cui MIQ (Quoziente Intellettivo di Macchina) sia di gran lunga più alto di quello dei sistemi convenzionali...". Oggi si usa il soft computing soprattutto nell'incrocio tra reti neurali e fuzzy logic si parla infatti di Neuro-fuzzy.

una prima reazione negativa nel mondo accademico questa nuova teoria negli anni a seguire registrò un crescente interesse. In Oriente, a riprova del fatto che queste ricerche avevano origini anche filosofiche, maturò un vivo fermento che contribuì in modo decisivo allo sviluppo della fuzzy logic. A Tokio T. Terano e H. Shibata e a Kanasai K. Tanaka e K. Asai diedero un positivo impulso alla teoria. Poco dopo nel 1974 in Gran Bretagna si inaugurava il primo sistema di controllo basato su logica Fuzzy si trattava di un generatore di vapore ideato da S. Assilian e E.H. Mamdani. Negli anni a seguire molte grandi industrie adottarono processi di produzione fuzzy. Tra i vari nomi si ricordano la Fuji Electric e la Hitachi. La ricerca fu sostenuta in Giappone da due grandi progetti, è sicuramente noto LIFE (Laboratory for International Fuzzy Engeneering Reserch). Intanto i campi di applicazione di tale teoria si espandevano e venivano usati dall'economia, alla sociologia per passare dall'ingegneria a all'informatica. In occidente interessanti sono stati i contributi anche del professore Bart Kosko che sviluppò il problema della non corrispondenza ossia l'interpretazione dei problemi come un chiaro scuro. Particolarmente apprezzata fu un'applicazione in campo finanziario usata per dare indicazioni sulla compravendita di titoli azionari che diede origine addirittura ad un fondo (Yamaichi Fuzzy Fund) operante sull'indice Nikkei Dow. La conferma della validità dell'idea è sottolineata dalla produzione di strumenti hardware e pacchetti software che fanno uso di tecniche fuzzy. Attualmente i chip così costruiti sono presenti nei sistemi di controllo della frenatura –ABS- così come in lavatrici. Nel box sono enunciate le più significative applicazioni.

LE BASI DELLA LOGICA

Io mento! In questa frase semplice e stringata si cela un paradosso. Applicando banalmente le regole della logica classica, si possono verificare due casi: che io menta o che io dica la verità. Nel primo dei due casi si rileva la contraddizione dovuta al fatto che se realmente mento non direi la verità dicendo che mento. Nel secondo caso la contraddizione è ancor più palese se dico la verità non posso affermare di mentire. In letteratura sono presenti un gran numero di simili paradossi, ma quello che solitamente viene riportato nel caso specifico è attribuito a Eubulide di Mileto. Si tratta del paradosso del mentitore, riscoperto da Bertrand Russel, il quale in forma sintetica viene enunciato come segue:

Il cretese Epimenide afferma che il cretese è bugiardo

Con un ragionamento analogo al precedente si intuisce che si tratta di un paradosso. Se la proposizione è vera il cretese Epimenide non risulterebbe bugiardo ma sincero, ma essendo cretese ciò non potrebbe essere. Se la proposizione è falsa il cretese non è bugiardo è quindi sincero in contraddizione con l'enunciazione di bugiardaggine. Insomma, pare che la soluzione a tale paradosso sia il valore complementare della proposizione, ossia v=!v. In altri termini la soluzione ossia il valore di verità della proposizione coincide con la sua negazione. Ma, come siamo abituati anche in programmazione, associato a vero il valore logico 1 e a falso il valore logico 0 e considerato che il complemento di 0 è 1 e viceversa, si ottiene che il risultato contraddittorio prima espresso v=!v che si può scrivere come v=1-v. Tale espressione non ha soluzione nella logica classica giacché v può assumere solo uno tra i due valori 0 e 1 ed entrambi non soddisfano l'equazione. Il valore che invece la soddisfa è v=1/2, che se per la logica bivalente è da scartare, ha un senso invece nella logica introdotta da Zadeh. Si tratta di una mezza verità o se preferite di una mezza falsità. Così si perviene ad una soluzione sfumata, sfuocata in cui si parla di grado di verità, nel caso specifico al 50%. Ovviamente, non vengono rispettati gli importanti pilastri che reggono la logica classica, ovvero il principio di non contraddizione e del terzo escluso. Si capisce anche perché per lo meno al principio tale teoria sia stata osteggiata, proprio perché minava alle fondamenta la logica classica.

DOVE SI USA LA LOGICA FUZZY

Di seguito sono riportati a titolo di pura enunciazione gli ambiti più conosciuti in cui è stata usata la fuzzy logic: ABS e cruise control, ossia sistema di frenatura; aria condizionata; macchine fotografiche digitali; elaborazione di immagini digitali; lavatrici; lavastoviglie; ascensori; riconoscimento di immagini (pattern ricognition). Ed ancora: come motore di molti giochi strategici come lord of ring, o come moderatore automatico di chat room ossia come un filtro di messaggi in funzione di un linguaggio.

DEFINIZIONI INSIEMI FUZZY

Negli studi a riguardo la logica classica o booleana, fondata cioè su due distinti valori, vero e falso è conosciuta come logica crisp, appunto chiara netta. Il primo passaggio per transitare alla logica fuzzy è quello di definire e generalizzare una funzione di appartenenza ad un insieme. Consideriamo un insieme A e un elemento x. Per capirci, in logica crisp la funzione di appartenenza ad un insieme

che indicheremo con g (in letteratura si trova anche con la lettera greca “mi”, di micro per intenderci) vale 1 se e soltanto se x appartiene ad A e 0 altrimenti, ossia se x non appartiene ad A. Quindi g(x,A) = 1 oppure g(x,A) = 0. Il valore della funzione viene detto grado di appartenenza e ha rilevanza nella logica fuzzy dove può assumere uno degli infiniti valori compresi nell’intervallo [0,1]. Quindi in un insieme fuzzy non si fa più il ragionamento per cui un elemento appartiene o no ad esso, ora ogni elemento sarà caratterizzato da un grado di appartenenza all’insieme stesso. Se si indicano i due valori più grande e più piccolo, xmax e xmin che possono appartenere all’insieme, attraverso una funzione matematica si può descrivere l’intera situazione di un insieme A e della relativa appartenenza dei valori ad esso. Nella figura 1a viene mostrato un esempio. In tale ottica cambiano gli operatori booleani usati come connettivi logici, che implicano anche il cambio di regole per la definizione delle operazioni tra insiemi: complemento, intersezione e unione. Imponendo alcune proprietà che questi operatori devono garantire, come la continuità e non decrescenza e la commutatività si ottengono con una dimostrazione matematica le funzioni di appartenenza:

- Per l’insieme B complemento – NOT A - : g(B,x)= 1-g(A,x)
- Per l’insieme C intersezione – A and B - : g(C,x)=min(g(A,x),g(B,x))
- Per l’insieme C unione – A or B - : g(C,x)=max(g(A,x),g(B,x))

Si dà una interpretazione grafica in **figura 1**.

UN ESEMPIO

Facciamo un esempio dando delle linee guida anche per una possibile implementazione. Consideriamo i due insiemi degli alti e dei vecchi. Per rimarcare ulteriormente la differenza con il metodo classico questi sarebbero descritti come di seguito. Avvaliamoci della chiara sintassi C++.

```
float altostandard (float x)
{   float al;
    if  (x<1.9) al=0
    else al = 1;
    return al }

int vecchiostandard  (int x)
{   int ve;
    if (x<80) ve=0
    else ve=1;
    return ve }
```

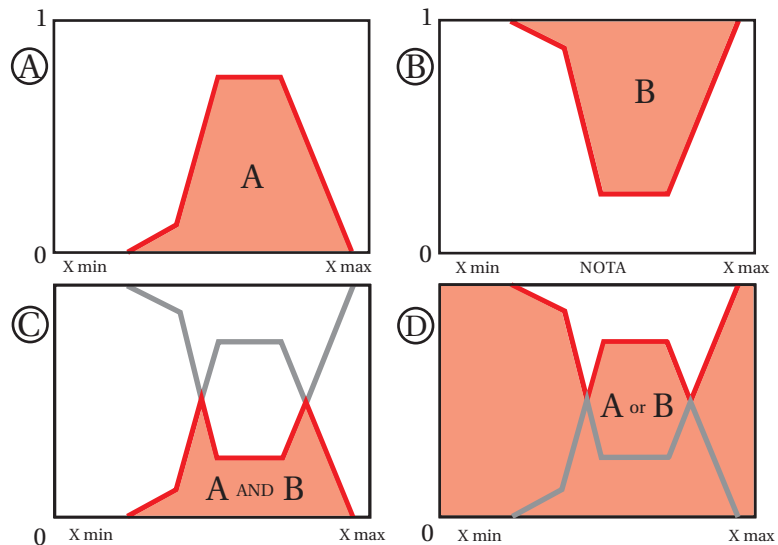



Fig. 1: ”a) Funzione di appartenenza per l’insieme A; b) Funzione di appartenenza per il complemento – NOT-; c) Funzione di appartenenza per l’insieme intersezione - A AND B -; d) Funzione di appartenenza per l’insieme unione – A OR B - ”

Se tali funzioni descrivono al meglio l’appartenenza di una persona all’insieme degli alti o dei vecchi, tale circostanza si riscontra se l’altezza e l’età di una persona superano rispettivamente il metro e novanta di altezza e gli ottanta anni di età. La stessa cosa non si può dire nel caso in cui non vi appartengano. Infatti, se ad esempio una persona non è alta saremmo indotti a pensare che sia bassa, ma con questo modello anche un corazziere di un metro e ottantotto risulterebbe basso. Stesso ragionamento per l’età. In **figura 2** è riportato l’ipotetica funzione di appartenenza dell’insieme vecchi per questa regola, il gradino dimostra che non si tratta di un approccio fuzzy.

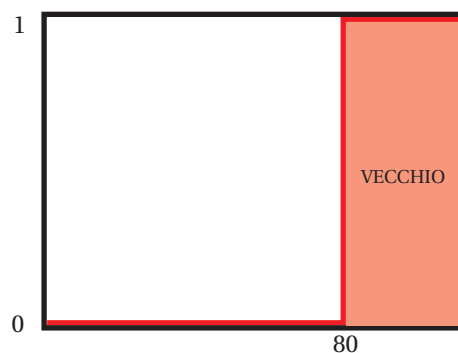



Fig. 2: ” Funzione di appartenenza all’insieme dei vecchi per un approccio classico”

Con la logica fuzzy si può introdurre un passaggio allo stato di alto o vecchio descritto da una funzione in modo da non avere le contraddizioni appena descritte. Per semplicità la funzione di passaggio è una semplice retta, ma modelli più sofisticati potrebbero usare funzioni più appropriate. Ecco come potrebbero essere opportunamente descritte le funzioni.

```
double altofuzzy (double x)
{   double al;
    if  (x<1.5) al=0;
    else if (x>=1.5) && (x<=1.9)  al=(x-1.5)*2.5
         else al = 1;
    return al }

int vecchiofuzzy  (int x)
{   int ve;
    if (x<80) ve=0;
    else if (x>=50) && (x<=80) ve=(x-50)/30
         else ve=1;
    return ve }
```

Si deve aggiungere una nuova condizione intermedia per la quale il valore assunto non è una costante bensì una funzione, nel caso specifico una retta. Si attiva quando il parametro x, altezza o età a seconda della funzione, è compreso in un intervallo. Nella tabella si può vedere come si ottengono in pratica i gradi di appartenenza all'insieme dei vecchi.

nome	età (anni)	g(VECCHIO,nome)
Costantino	2	0
Fabio	36	0
Gino	59	0.3
Antonio	75	0.83
Pippo	90	1

In **figura 3** è invece sotto osservazione l'insieme degli alti e il suo complemento, non alti, che questa volta possiamo con più tranquillità definire bassi. Siano AEV, AOV e NA rispettivamente gli insiemi degli alti e vecchi, degli alti o vecchi e dei non alti. Applicando gli operatori fuzzy prima descritti si possono ottenere i seguenti risultati riportati in tabella:

nome	altezza (m)	età (anni)	g(ALTO,nome)	g(VECCHIO,nome)	AEV	AOV	NA
Pippo	1.5	90	0	1	0	1	1
Nino	1	3	0	0	0	0	1
Ugo	1.6	30	0.25	0	0	0.25	0.75
Nico	1.7	55	0.5	0.17	0.17	0.5	0.5
Rocco	1.8	65	0.75	0.5	0.5	0.75	0.25
Piero	1.88	78	0.95	0.93	0.93	0.95	0.05
Ciccio	2	52	1	0.07	0.07	1	0

calcolo aspetti	Calcolo classico	Soft computing
Programmazione	Rigida	Addestrabile e adattativo
Logica	Basato su logica binaria	Basata su valori multipli
Approccio	Deterministico	Probabilistico, fuzzy
Dati	Esatti	Incerti, ambigui, incompleti
Operazioni	Serie	Parallelo
Risultati	Precisi	Approssimati

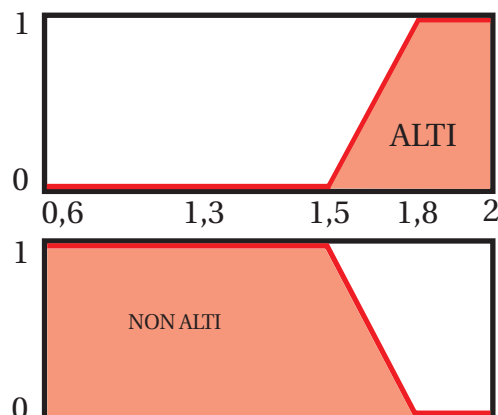



Fig. 3: "a) Funzione di appartenenza all'insieme degli Alti; b) Funzione di appartenenza all'insieme dei non alti

ALTRI OPERATORI

Oltre ai connettivi logici spesso sono richiesti altri operatori. Ad esempio, può essere necessario il calcolo del prodotto tra due funzioni di appartenenza, rispetto a due insiemi diversi (A e B), in tal caso sono stati introdotti a secondo delle applicazioni altri operatori di congiunzione, di seguito ne sono riportati alcuni:

- Prodotto algebrico: g(A,x)*g(B,x)
- Prodotto limitato: max{0, g(A,x)+g(B,x)-1}
- Prodotto drastico: - g(A,x) se g(N,x) =1
  - g(N,x) se g(S,x) =1
  - 0 se g(A,x),g(N,x)<1

Dove g(N,x) è un valore normalizzato
Così come sono presenti altri operatori di unione o somma:

- La somma algebrica: g(A,x)+g(N,x)- g(A,x)*g(B,x)
- La somma limitata: min{1, g(A,x)+g(B,x)}

CONCLUSIONI

Ecco un confronto tra calcolo tradizionale e calcolo soft:
Il monito di Zadeh sta proprio nella difficoltà futura che l'uomo avrà a trattare una sempre maggiore potenza di elaborazione a fronte di problemi sempre più difficili da formalizzare e intrisi di imprecisioni non facilmente eliminabili. Il calcolo soft quindi giocherà un ruolo da protagonista. Ad oggi assistiamo ad un sempre maggiore interesse verso queste tematiche anche se a dire il vero non hanno "sfondato" come qualcuno qualche anno fa prevedeva. Evidentemente la teoria deve ancora maturare e devono essere sviluppate ulteriori applicazioni. Per quanto ci riguarda ci piace osservarne gli sviluppi ed essere eventualmente tra i pionieri delle loro applicazioni.

Fabio Grimaldi